

1. [Structural Computational Biology: Introduction and Background](#)
2. [Representing Proteins in Silico and Protein Forward Kinematics](#)
3. [Protein Inverse Kinematics and the Loop Closure Problem](#)
4. [Molecular Shapes and Surfaces](#)
5. [Molecular Distance Measures](#)
6. [Protein Classification, Local Alignment, and Motifs](#)
7. [Dimensionality Reduction Methods for Molecular Motion](#)
8. [Robotic Path Planning and Protein Modeling](#)
9. [Motion Planning for Proteins: Biophysics and Applications](#)
10. [Protein-Ligand Docking, Including Flexible Receptor-Flexible Ligand Docking](#)
11. Homework assignments
  1. [Assignment 1: Visualization and Ranking of Protein Conformations](#)
  2. [Assignment 2: Performing Rotations](#)
  3. [Assignment 3: Inverse Kinematics](#)

## Structural Computational Biology: Introduction and Background

### Topics in this Module

- [Proteins and Their Significance to Biology and Medicine](#)
- [Protein Structure](#)
- [Experimental Methods for Protein Structure Determination](#)
- [Protein Structure Repositories](#)
- [Visualizing Protein Structures](#)

### Proteins and Their Significance to Biology and Medicine

Proteins are the molecular workhorses of all known biological systems. Among other functions, they are the motors that cause muscle contraction, the catalysts that drive life-sustaining chemical processes, and the molecules that hold cells together to form tissues and organs.

The following is a list of a few of the diverse biological processes mediated by proteins:

- Proteins called enzymes catalyse vital reactions, such as those involved in metabolism, cellular reproduction, and gene expression.
- Regulatory proteins control the location and timing of gene expression.
- Cytokines, hormones, and other signalling proteins transmit information between cells.
- Immune system proteins recognize and tag foreign material for attack and removal.
- Structural proteins prevent cells from collapsing on themselves, as well as forming large structures such as hair, nails, and the protective, largely impermeable outer layer of skin. They also provide a framework along which molecules can be transported within cells.

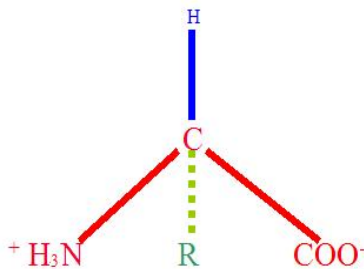
The estimate of the number of genes in the human genome has been changing dramatically since it was annotated (the latest gene count estimates can be found in this [Wikipedia article on the human genome](#)). Each gene encodes one or more distinct proteins. The total number of distinct proteins in the human body is larger than the number of genes due to [alternate splicing](#). Of those, only a small fraction have been isolated and

studied to the point that their purpose and mechanism of activity is well understood. If the functions and relationships between every protein were fully understood, we would most likely have a much better understanding of how our bodies work and what goes wrong in diseases such as cancer, amyotrophic lateral sclerosis, Parkinson's, heart disease and many others. As a result, protein science is a very active field. As the field has progressed, computer-aided modeling and simulation of proteins have found their place among the methods available to researchers.

## Protein Structure

An amino acid is a simple organic molecule consisting of a basic (hydrogen-accepting), amine group bound to an acidic (hydrogen-donating) carboxyl group via a single intermediate carbon atom:

An  $\alpha$ -amino acid

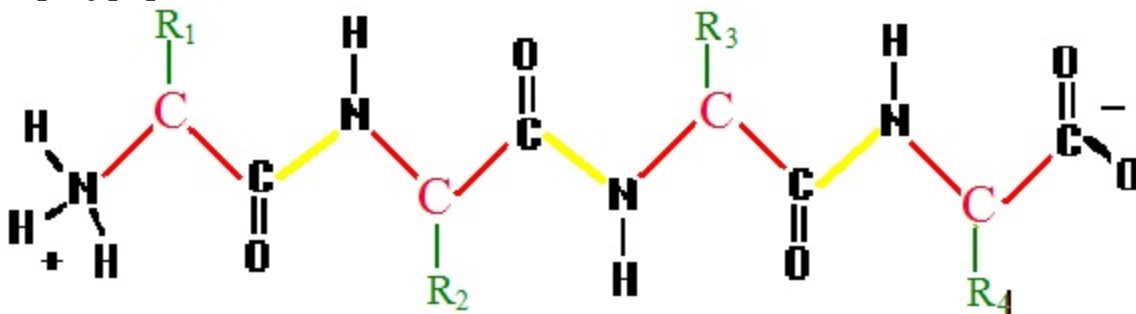


A generic  $\alpha$ -amino acid. The "R" group is variable, and is the only difference between the 20 common amino acids. This form is called a zwitterion, because it has both positive and negatively charged atoms. The zwitterionic state results from the

amine group (NH<sub>2</sub>)  
gaining a hydrogen  
atom from solution,  
and the acidic  
group (COO)  
losing one.

During the translation of a gene into a protein, the protein is formed by the sequential joining of amino acids end-to-end to form a long chain-like molecule, or **polymer**. A polymer of amino acids is often referred to as a **polypeptide**. The genome is capable of coding for 20 different amino acids whose chemical properties depend on the composition of their **side chains** ("R" in the above figure). Thus, to a first approximation, a protein is nothing more than a sequence of these amino acids (or, more properly, amino acid **residues**, because both the amine and acid groups lose their acid/base properties when they are part of a polypeptide). This sequence is called the **primary structure** of the protein.

A polypeptide

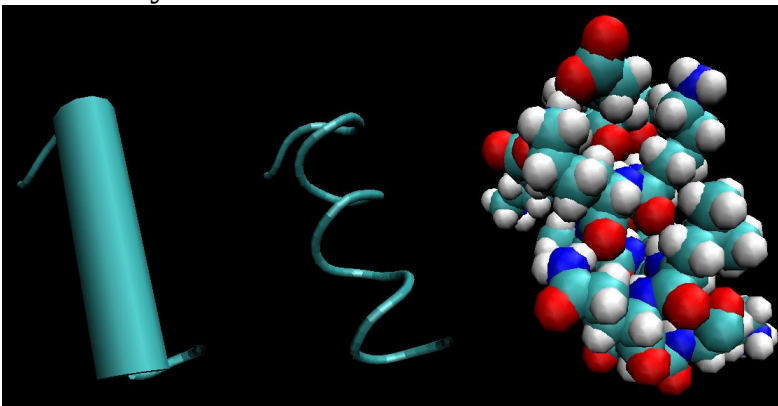


A generic polypeptide chain. The bonds shown in yellow, which connect separate amino acid residues, are called **peptide bonds**.

The [Wikipedia entry on amino acids](#) provides a more detailed background, including the structure, properties, abbreviations, and genetic codes for each of the 20 common amino acids.

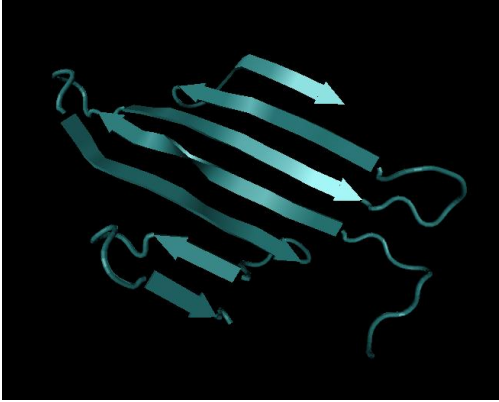
The primary structure of a protein is easily obtainable from its corresponding gene sequence, as well as by experimental manipulation. Unfortunately, the primary structure is only indirectly related to the protein's function. In order to work properly, a protein must fold to form a specific three-dimensional shape, called its **native structure** or **native conformation**. The three-dimensional structure of a protein is usually understood in a hierarchical manner. **Secondary structure** refers to folding in a small part of the protein that forms a characteristic shape. The most common secondary structure elements are  **$\alpha$ -helices** and  **$\beta$ -sheets**, one or both of which are present in almost all natural proteins.

#### Secondary Structure: $\alpha$ -helix



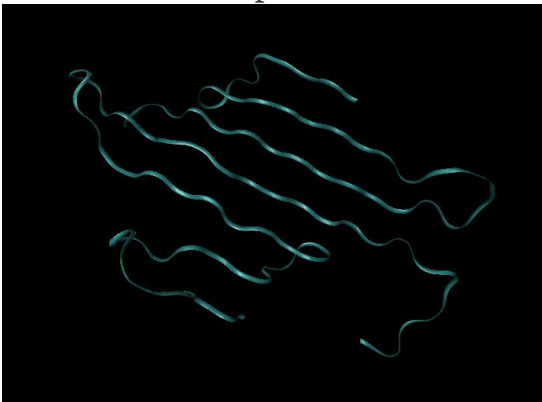
$\alpha$ -helices, rendered three different ways. Left is a typical cartoon rendering, in which the helix is depicted as a cylinder. Center shows a trace of the backbone of the protein. Right shows a space-filling model of the helix, and is the only rendering that shows all atoms (including those on side chains).

#### Secondary Structure: $\beta$ -sheet Cartoon representation



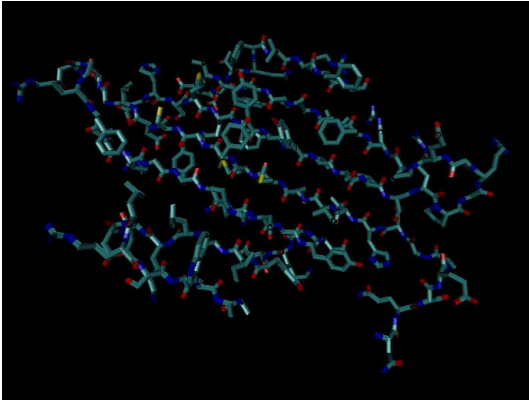
Different parts of the polypeptide strand align with each other to form a  $\beta$ -sheet. This  $\beta$ -sheet is **anti-parallel**, because adjacent segments of the protein run in opposite directions.

Ribbon representation



$\beta$ -sheets are sometimes referred to as  $\beta$  pleated sheets, because of the regular zig-zag of the strands evident in this representation.

### Bond representation



Each segment in this representation represents a bond. Unlike the other two representations, side chains are illustrated. Note the alignment of oxygen atoms (red) toward nitrogen atoms (blue) on adjacent strands. This alignment is due to hydrogen bonding, the primary interaction involved in stabilizing secondary structure.

Beta-sheets represented in three different rendering modes: cartoon, ribbon, and bond representations.

**Tertiary structure** refers to structural elements formed by bringing more distant parts of a chain together into structural **domains**. The spatial

arrangement of these domains with respect to each other is also considered part of the tertiary structure. Finally, many proteins consist of more than one polypeptide folded together, and the spatial relationship between these separate polypeptide chains is called the **quaternary structure**. It is important to note that the native conformation of a protein is a direct consequence of its primary sequence and its chemical environment, which for most proteins is either aqueous solution with a biological pH (roughly neutral) or the oily interior of a cell membrane. Nevertheless, no reliable computational method exists to predict the native structure from the amino acid sequence, and this is a topic of ongoing research. Thus, in order to find the native structure of a protein, experimental techniques are deployed. The most common approaches are outlined in the next section.

## Experimental Methods for Protein Structure Determination

A **structure** of a protein is a three-dimensional arrangement of the atoms such that the integrity of the molecule (its connectivity) is maintained. The goal of a protein structure determination experiment is to find a set of three-dimensional (x, y, z) coordinates for each atom of the molecule in some natural state. Of particular interest is the native structure, that is, the structure assumed by the protein under its biological conditions, as well as structures assumed by the protein when in the process of interacting with other molecules. Brief sketches of the major structure determination methods follow:

### X-ray Crystallography

The most commonly used and usually highest-resolution method of structure determination is **x-ray crystallography**. To obtain structures by this method, laboratory biochemists obtain a very pure, crystalline sample of a protein. X-rays are then passed through the sample, in which they are diffracted by the electrons of each atom of the protein. The diffraction pattern is recorded, and can be used to reconstruct the three-dimensional pattern of electron density, and therefore, within some error, the location of each atom. A high-resolution **crystal structure** has a resolution on the

order of 1 to 2 **Angstroms** (Å). One Angstrom is the diameter of a hydrogen atom ( $10^{-10}$  meter, or one hundred-millionth of a centimeter).

Unlike other structure determination methods, with x-ray crystallography, there is no fundamental limit on the size of the molecule or complex to be studied. However, in order for the method to work, a pure, crystalline sample of the protein must be obtained. For many proteins, including many membrane-bound receptors, this is not possible. In addition, a single x-ray diffraction experiment provides only static information - that is, it provides only information about the native structure of the protein under the particular experimental conditions used. As we will see later, proteins are often flexible, dynamic objects when in their natural state in solution, so a single structure, while useful, may not tell the full story. More information on X-ray Crystallography is available at [Crystallography 101](#) and in the [Wikipedia](#).

## NMR

**Nuclear Magnetic Resonance (NMR)** spectroscopy has recently come into its own as a protein structure determination method. In an NMR experiment, a very strong magnetic field is transiently applied to a sample of the protein being studied, forcing any magnetic atomic nuclei into alignment. The signal given off by a nucleus as it returns to an unaligned state is characteristic of its chemical environment. Information about the atoms within two chemical bonds of the resonating nucleus can be deduced, and, more importantly, information about which atoms are spatially near each other can also be found. The latter information leads to a large system of distance constraints between the atoms of the protein, which can then be solved to find a three-dimensional structure. Resolution of NMR structures is variable and depends strongly on the flexibility of the protein. Because NMR is performed on proteins in solution, they are free to undergo spatial rearrangements, so for flexible parts of the protein, there may be many more than one detectable structures. In fact, NMR structures are generally reported as **ensembles** of 20-50 distinct structures. This makes NMR the only structure determination technique suited to elucidating the behavior of **intrinsically unstructured proteins**, that is, proteins that lack a well-

defined tertiary structure. The reported ensemble may also provide insight into the dynamics of the protein, that is, the ways in which it tends to move.

NMR structure determination is generally limited to proteins smaller than 25-30 kilodaltons (kDa), because the signals from different atoms start to overlap and become difficult to resolve in that range. Additionally, the proteins must be soluble in concentrations of 0.2-0.5 mM without aggregation or precipitation. For more information on how NMR is used to find molecular structures, please see [NMR Basics](#) and [The World of NMR: Magnets, Radio Waves, and Detective Work](#) at the National Institutes of Health's [The Structures of Life](#) website.

## **Electron Diffraction**

**Electron diffraction** works under the same principle as x-ray crystallography, but instead of x-rays, electrons are used to probe the structure. Because of difficulties in obtaining and interpreting electron diffraction data, it is rarely used for protein structure determination. Nevertheless, ED structures do exist in the PDB. For more on ED, see this [Wikipedia article](#).

## **Structure Prediction of Large Complexes**

Large macromolecular complexes and molecular machines present a particular challenge in structure determination. Generally too large to be crystallized, and too complex to solve by NMR, determining the structure of these objects usually requires the combination of high-resolution microscopy combined with computational refinement and analysis. The main techniques used are [cryo-electron microscopy \(Cryo-EM\)](#) and standard light microscopy.

## **Protein Structure Repositories**

Most of the protein structures discovered to date can be found in a large protein repository called the [RCSB Protein DataBank \(PDB\)](#). The **Protein Data Bank (PDB)** is a public domain repository that contains experimentally determined structures of three-dimensional proteins. The majority of the proteins in the PDB have been determined by x-ray crystallography, but the number of proteins determined using NMR methods has been increasing as efficient computational techniques to derive structures from NMR data have been developed. A few electron diffraction structures are also available. The PDB was originally established at Brookhaven National Laboratory in October, 1971, with 7 structures. Currently, the database is maintained by Rutgers University, the State University of New Jersey, the San Diego Supercomputer Center at the University of California, San Diego, and the National Institute of Standards and Technology. The current number of proteins (and/or nucleic acids) in the PDB database is displayed at the top-right corner of the main PDB page. The imaging method statistics of these structures (i.e., which methods were used for what fraction of the structures), as well as other classifications, can be found [here](#). The European Bioinformatics Institute Macromolecular Structure Database group (UK) and the Institute for Protein Research at Osaka University (Japan) are international contributors to the contents of the PDB.

## Visualizing Protein Structures

Numerous tools are available for visualizing the structures stored in the PDB and other repositories. Most such tools allow a detailed examination of the molecule in a variety of rendering modes. For example, sometimes it may be useful to have a detailed image of the surface of the molecule as experienced by a molecule of water. For other purposes, a simple, cartoonish representation of the major structural features may be sufficient.

### A Few Molecular Visualization Programs

- [Visual Molecular Dynamics \(VMD\)](#) was originally developed for viewing molecular simulation trajectories. It is a very powerful, full-featured, and customizable molecular viewing package. Customization

is available using Tcl/Tk scripting. Information on Tcl/Tk scripting can be found at this [Tcl/Tk](#) website.

- [PyMol](#) is an open-source molecular viewer that can be used to generate professional-looking images. PyMol is highly customizable through the Python scripting language.
- [Protein Explorer](#) is an easy-to-use, web browser-based visualization tool. Protein explorer is built using the [MDL Chime](#) browser plugin, which in turn is based on the [RasMol](#) viewer. Because Chime only works under Windows and Macintosh OS, the use of Protein Explorer is restricted to those platforms.
- [JMol](#) is a Java-based molecular viewer. In applet form, it can be downloaded on-the-fly to view structures from the web. A stand-alone version also exists, which can be used independently of a web browser.
- [Chimera](#) is a powerful visualizer and analysis tool that can be comfortably used with very large molecular complexes. It can also produce very high-quality images for use in presentations and publications.

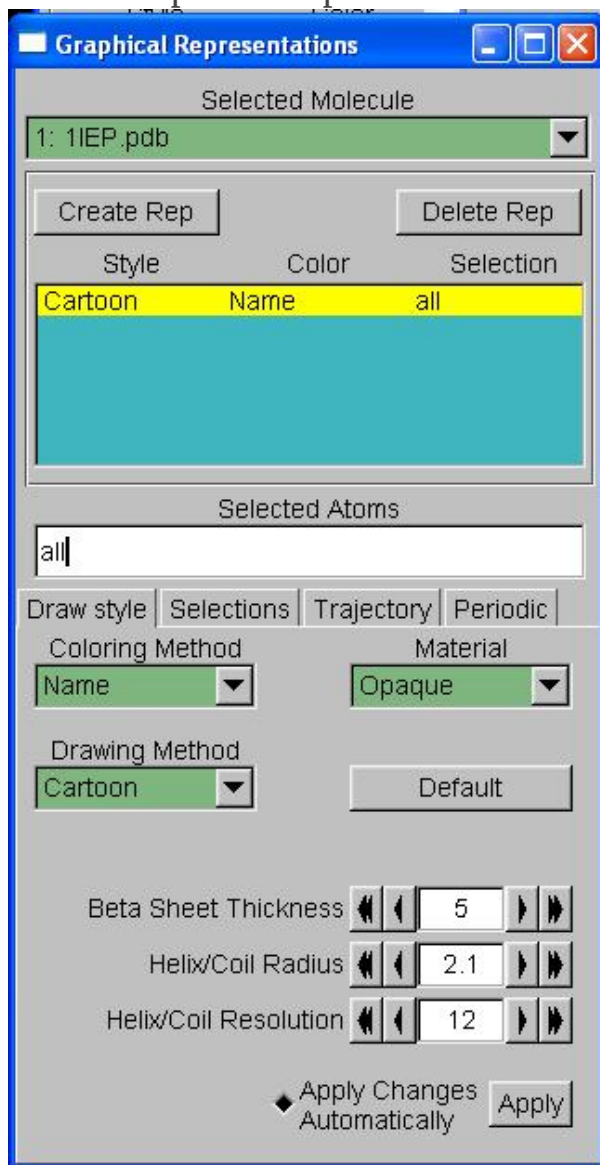
## Visualizing HLA-AW with VMD

What follows will be a very brief introduction to what can be done with VMD. Only the most basic viewing functionality will be discussed. For a complete description of the capabilities of VMD and how to use them, please refer to the [VMD web site](#).

In this section, a human leukocyte-associated antigen, HLA-AW (PDB structure ID 2HLA), will be shown under various rendering methods in VMD. This section is intended to convey, first, a general idea of the types of visual representations that are available for protein structures, and second, what information is and is not conveyed by each representation.

VMD allows the user to load and view molecule description files in a wide variety of common formats, including trajectory files with multiple structures of the same molecule, such as might be generated by a simulation. Once the molecules are loaded, the way each molecule is rendered may be controlled using the Graphical Representations menu:

## VMD Graphical Representations menu



This menu allows the user to control in detail how each molecule is rendered.

## VMD atom coloring methods

Name
Type
Element
ResName
ResType
ResID
Chain
SegName
Molecule
Structure
ColorID
Beta
Occupancy
Mass
Charge
Pos
User
Index
Backbone
Throb
Timestep
Volume

Coloring  
schemes  
to  
highlight  
features  
of  
interest.

VMD molecule drawing methods

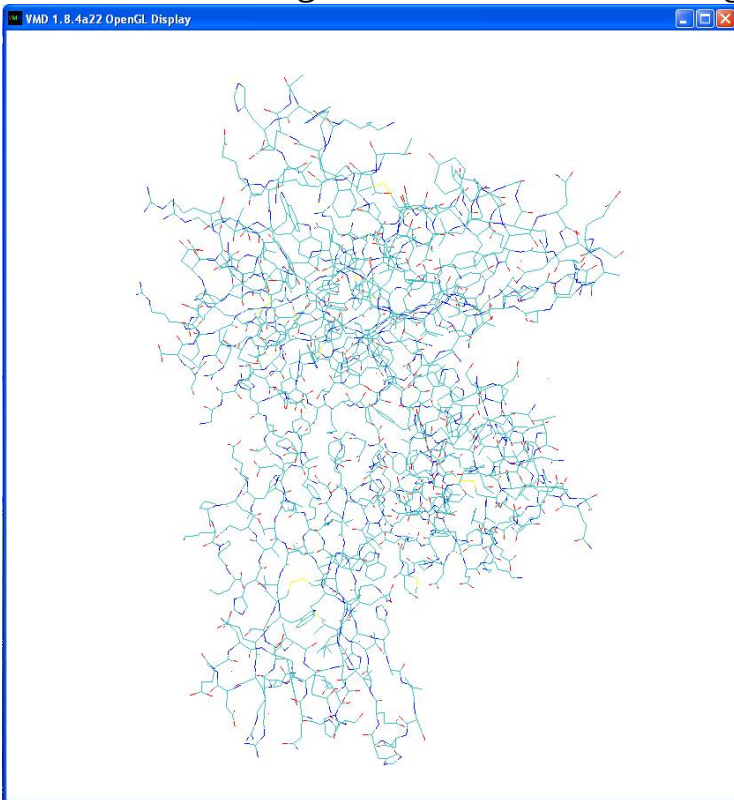
Lines  
Bonds  
DynamicBonds  
HBonds  
Points  
VDW  
CPK  
Licorice  
Trace  
Tube  
Ribbons  
NewRibbons  
Cartoon  
NewCartoon  
MSMS  
Surf  
VolumeSlice  
Isosurface  
Beads  
Dotted  
Solvent

Rendering methods  
in VMD.

Which  
one to use  
depends  
on the  
features  
to  
highlight.

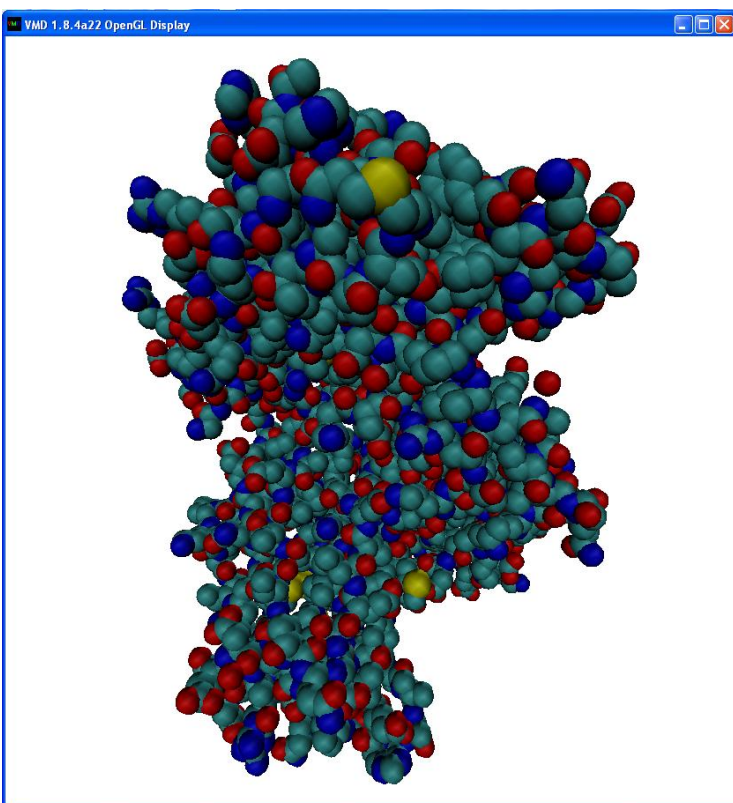
The built-in rendering options of  
VMD.

Molecules may be displayed by various rendering modes:  
HLA-AW. Drawing method: LINES. Coloring method: NAME



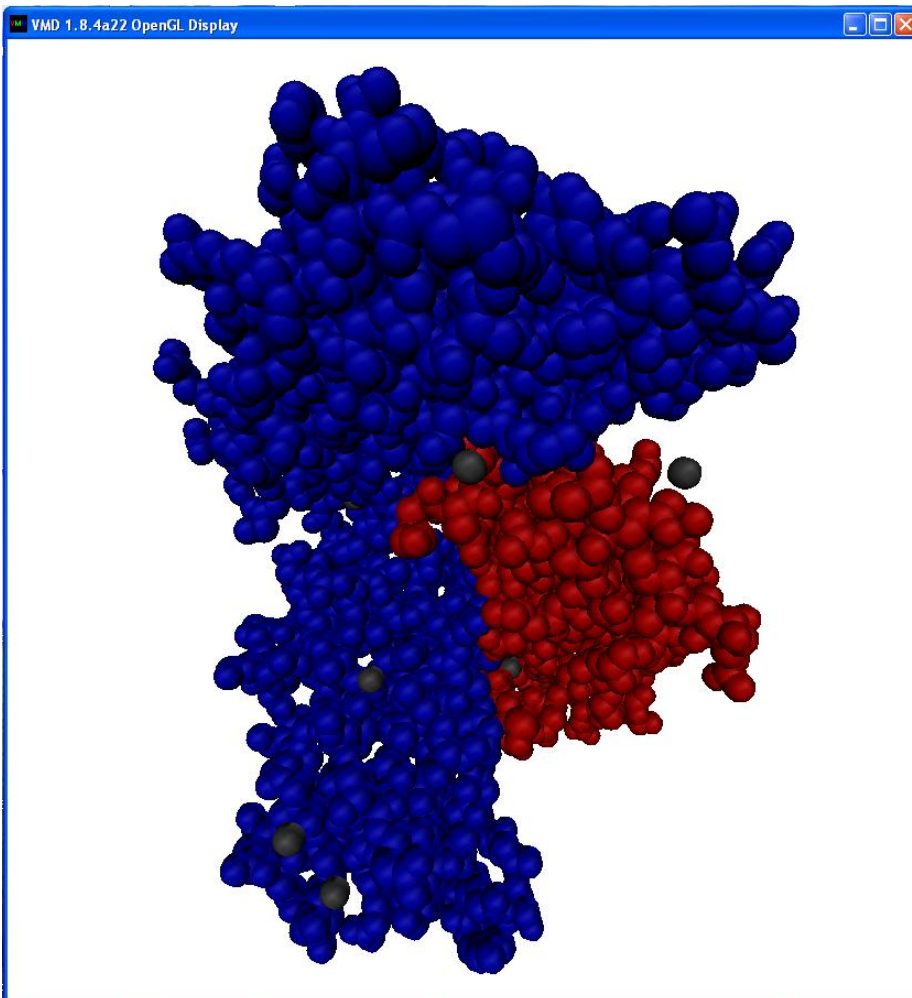
In this representation, each line represents a bond between two atoms. The color of each half-bond corresponds to the element of the atom at the corresponding end of the bond (red for oxygen, blue for nitrogen, yellow for sulfur, and teal for carbon). Line representation gives a clear idea of the molecule's connectivity, but for large molecules it can be difficult to isolate protein sub-structures.

HLA-AW. Drawing method: VDW. Coloring method: NAME



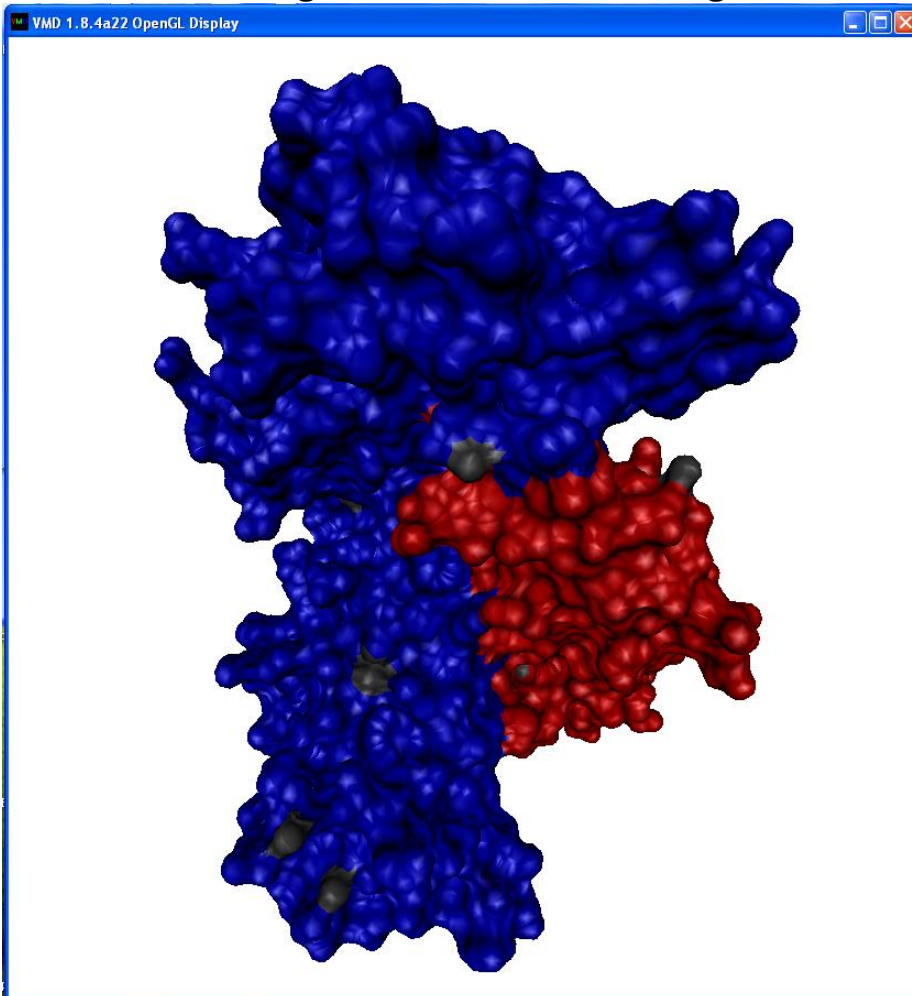
Here each atom is represented by a sphere whose radius is the **Van der Waals radius** of the atom. The Van der Waals radius is half the separation of unbonded atoms packed as tightly as possible, and provides a rough notion of a collision radius, although it is not a firm barrier. This representation of the molecule gives a rough sense of its shape, and is sometimes called a **space-filling** model.

HLA-AW. Drawing method: VDW. Coloring method: CHAIN



This rendering is the same as in the previous figure, except that now the atoms are colored based on which polypeptide chain they belong to. HLA-AW consists of two chains, the alpha chain (blue), which folds into three domains and the smaller  $\beta 2$  microglobulin (red), which is a component of a whole class of HLA proteins. Coloring by chain allows an inspection of how the polypeptide subunits come together to form the whole quaternary structure of the protein. The black balls are water molecules near the surface of the protein that always appear in the same place in crystal structures, and may therefore be considered part of the structure for some applications.

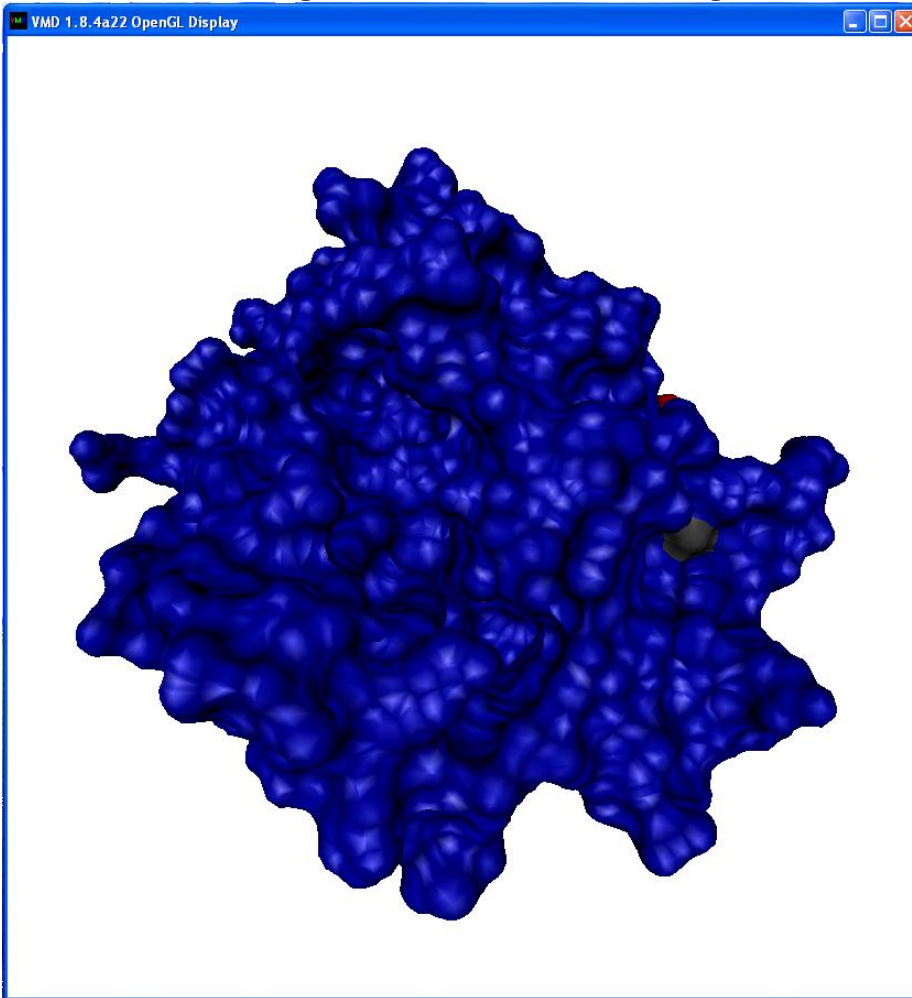
HLA-AW. Drawing method: SURF. Coloring method: CHAIN



The Surf drawing mode renders a surface swept out by a sphere of some set size skimming the protein.

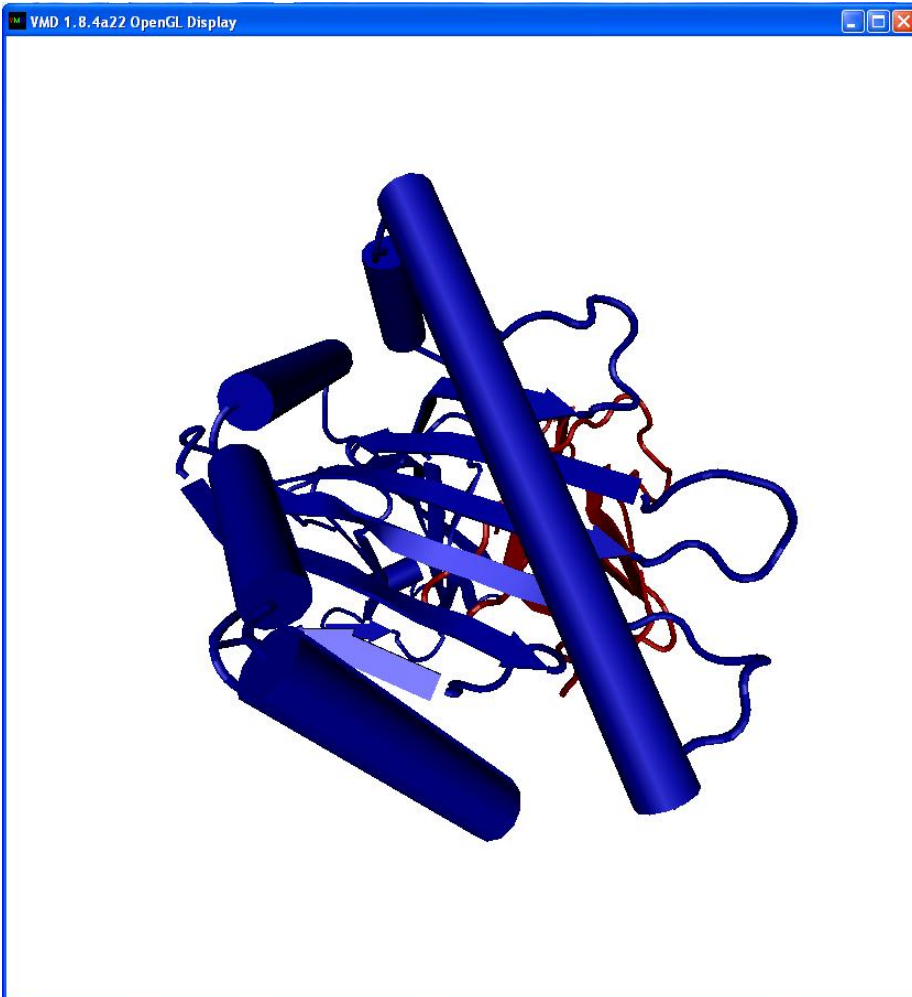
Usually, this size is approximately that of a water molecule, in which case the rendered surface is very similar to the **solvent-accessible surface**. Note that it is impossible to deduce the connectivity of the atoms from this image or from the space filling image in the previous figure. Overall shape, rather than connectivity, is the information conveyed by these representations. Hence, both backbone-based and surface-based renderings are necessary to fully understand a protein's structure.

HLA-AW. Drawing method: SURF. Coloring method: CHAIN



Here the protein has been rotated approximately 90 degrees toward the viewer, so that, compared to the previous image, we are looking down from above. The deep groove running from the top left to lower right is the **binding pocket** of the protein.

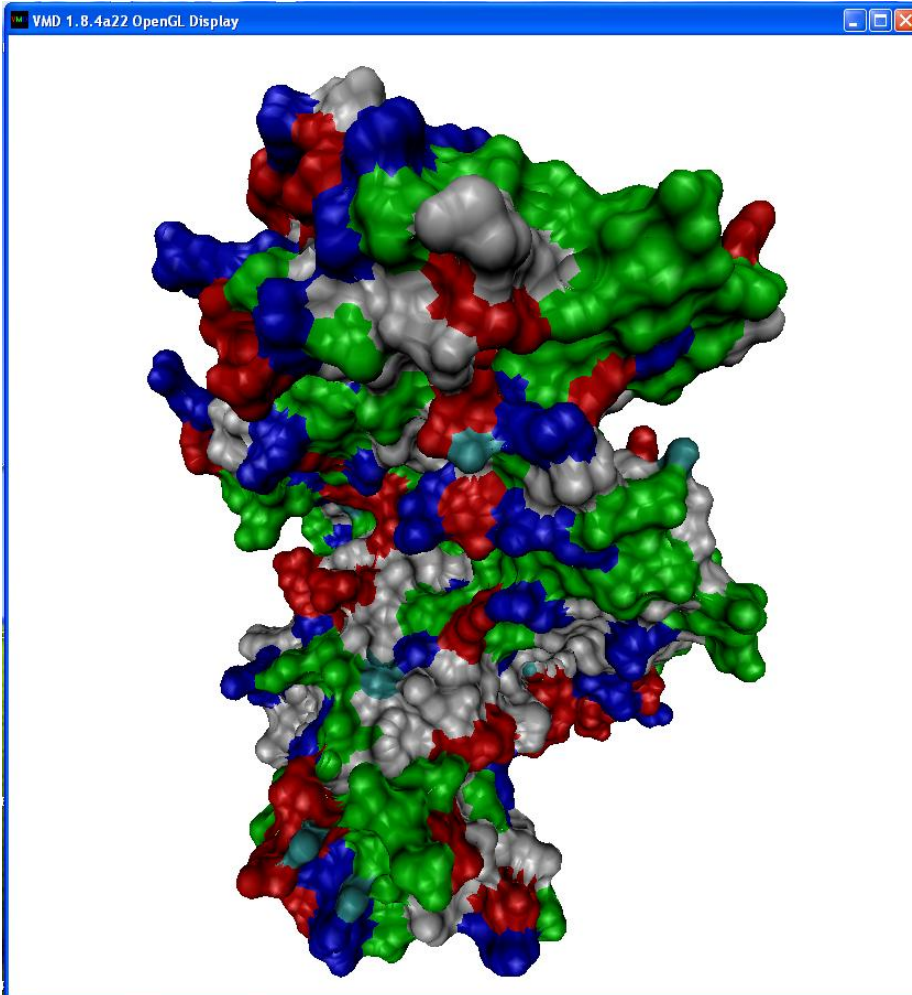
HLA-AW. Drawing method: CARTOON. Coloring method: CHAIN



Cartoon rendering places an emphasis on secondary structure. Beta sheets appear as flattened arrows, and alpha helices appear as cylinders. These are common conventions in representing protein secondary structure. By examining this image, we can see that the walls of the binding pocket observed in the previous figure consist of alpha helices, and the floor is an **anti-parallel** beta sheet. In anti-parallel beta sheets, adjacent strands run in the opposite direction (notice the arrow points alternate in direction). Note that this representation only conveys information about the backbone connectivity of the protein. Side chain atoms are

omitted, and therefore the overall shape is only a very coarse approximation.

HLA-AW. Drawing method: SURF. Coloring method: RESTYPE



Alternative coloring methods can provide additional insight into a protein's structure and function. Here each atom is colored based on whether the side chain of the amino acid residue to which it belongs is acidic (red), basic (blue), polar neutral (green), or apolar (gray). Note that residues on the surface of the protein tend to be hydrophilic (attracted to water, in red, blue, and green), whereas residues closer to the core of the protein tend to be hydrophobic (greasy or water repellant, in gray). This is

characteristic of proteins that exist in aqueous solution in nature. Their native structure is stabilized by a tendency for the hydrophilic residues to interact with the solvent water molecules, while the hydrophobic residues are driven together away from the solvent. Clusters of hydrophobic residues on the surface often indicate a location that is usually protected from solvent in the natural state, either by interaction with another molecule or by part of the protein itself.

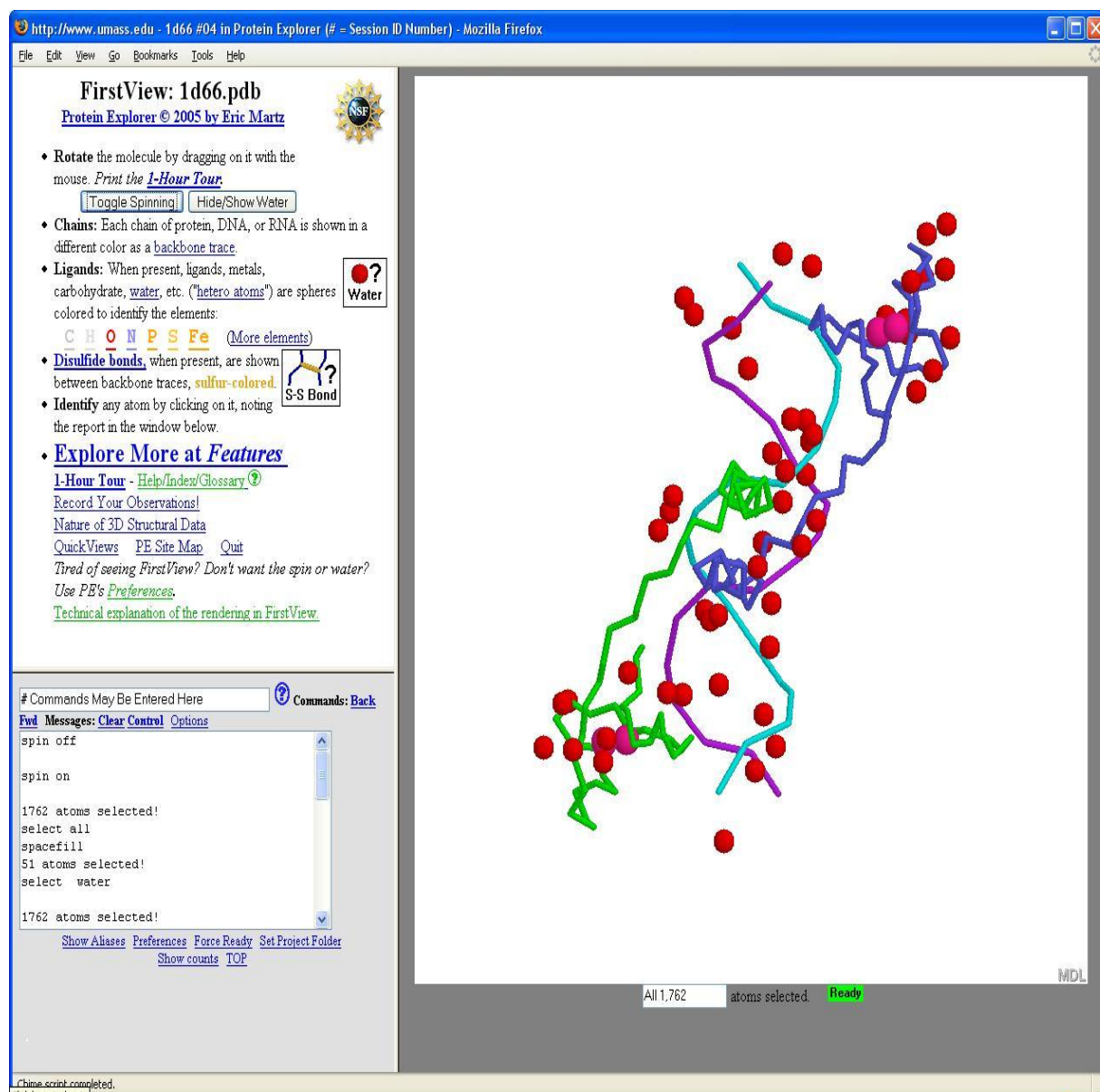
## **Visualizing HLA-AW with Protein Explorer**

Protein Explorer is designed as a user-friendly but fairly full-featured visualizer. It is not as scriptable or as powerful as some other visualizers such as VMD and PyMol, but it is one of the quickest and easiest to get started with. It is used through a web browser, either by accessing it through the [Protein Explorer website](#) (via the Quick-Start Protein Explorer link), or as an offline version, downloadable from [this page](#). Both versions require the MDL Chime molecular viewing plugin, which you can download from [here](#) (registration required).

As with VMD above, a human leukocyte-associated antigen, HLA-AW (PDB structure ID 2HLA), will be shown in various renditions.

Upon opening, Protein Explorer will load a default molecule and display it (this feature may be disabled via a setting under "preferences" in the lower left frame):

**Protein Explorer at Startup**



The interface contains three areas. The frame on the right contains the rendering window, where the molecule is displayed. The lower left frame contains an input box for text commands and a text box that displays general text output from the program: What commands have been executed, what the program is currently doing, etc. The top left frame generally contains the user interface in the form of buttons and links. Its exact contents vary with use.

Clicking on the "PE Site Map" link pops up a window containing Protein Explorer's top-level menu:

### Protein Explorer Site Map Window



Each option contains a helpful tooltip which can be seen by hovering the mouse cursor over it.

"New Molecule" allows the user to load a molecule either directly from the PDB or from the local filesystem. "Reset Session" returns to the default view and rendering style, which can be a useful shortcut. "Quick Views" opens up a menu from which the

user can select how  
the molecule is  
rendered.

Once a molecule is loaded, the "Quick Views" menu allows the user to control how it is displayed:

Protein Explorer QuickViews Interface

**QuickViews: 2HLA.pdb.gz**

[Protein Explorer © 2005 by Eric Martz](#)

• **SELECT**      • **DISPLAY**      **COLOR**

1.  2.  3.

**Beyond QuickViews: [PE Site Map](#)**

---

*QuickViews **Boolean** / Size Options*

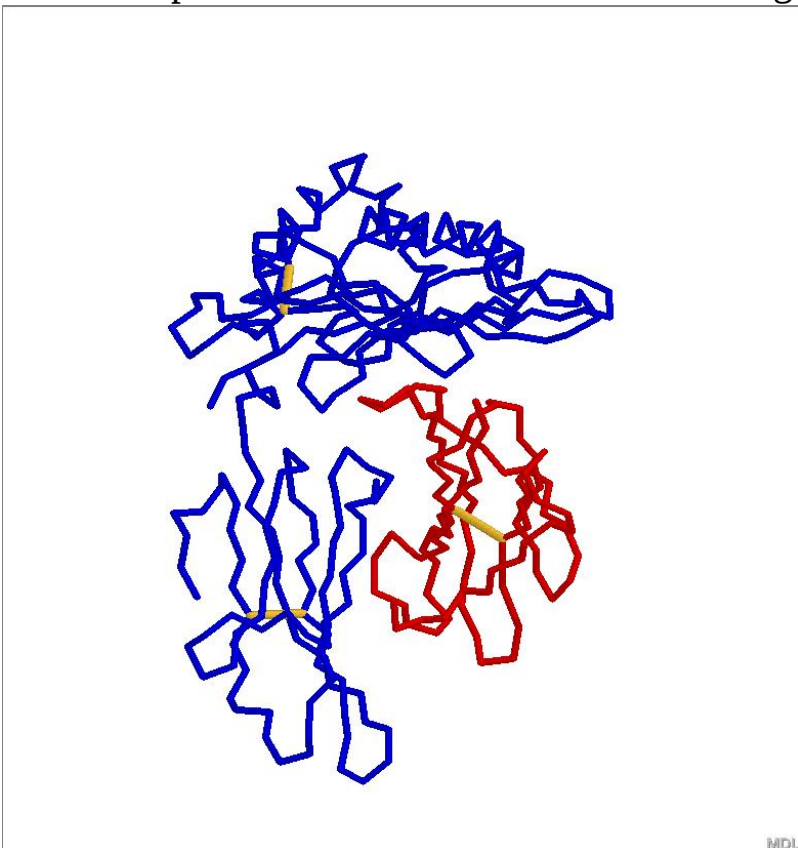
- ♦ New **Selection**  previous selection.
- ♦ New **Display**  previous display for the selected atoms.  
Clicking on the symbol to the left of the words SELECT or DISPLAY above the main menus steps through the options listed just above.
- ♦ Thickness of **Backbones and Traces**:
- (Thicknesses are given as radii in Angstroms. "SS" = Secondary Structure. "Temp" = Temperature. SS and Temp work with traces only)
- ♦ Thickness of **Sticks** (wireframe):
- ♦ Radii of **Spacefill** spheres:
- "Ball+Stick" uses spheres of radius 0.45 Angstroms, and sticks of radius 0.15 Angstroms.

\* Defaults.

The "SELECT" pulldown menu allows the user to pick a group of atoms based on their properties, their location, the structural elements in which

they are involved, or by directly clicking them. The "DISPLAY" pulldown menu then allows the user to determine the style in which the selected atoms are rendered. Most of the styles available through VMD are also available in Protein Explorer. The "COLOR" pulldown menu allows the user to determine how the atoms are colored. Options include coloring by secondary structure elements, atom type, subunit (chain), a spectrum from end to end of the protein, and by properties such as charge and polarity.

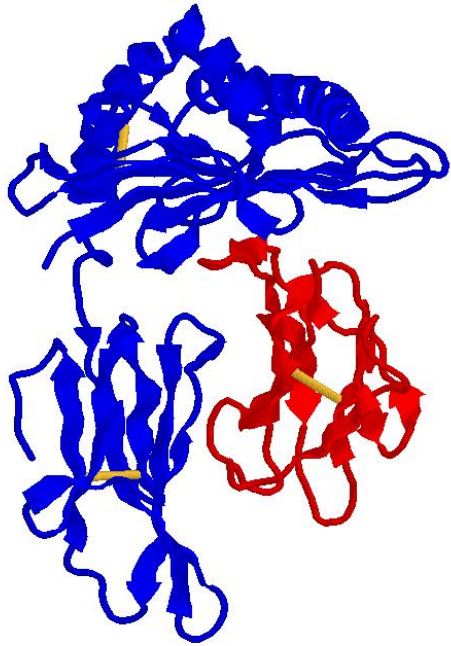
### Protein Explorer: HLA-AW Backbone Rendering



This rendering mode shows the protein backbone (no side chains) through the alpha carbons of each amino acid residue. It gives

the user a sense of how the chains fold to form the structure, but not it's full shape, since all side chain atoms have been removed. The yellow bars are disulfide bonds, which are covalent bonds that lock distant parts of the chain together to help maintain the structure.

### Protein Explorer: HLA-AW Cartoon Style



MDL

Cartoon rendering works as for VMD. As in the backbone rendering above, side chains are ignored, and the protein backbone is rendered as a smoothly curving tube. Beta sheets appear as flattened arrows, and alpha helices appear as spiraling ribbons.

## Protein Explorer Advanced Explorer Menu

### Advanced Explorer: 2HLA.pdb.gz ?

[Protein Explorer © 2005 by Eric Martz](#)

1. [Surfaces](#),
2. [Contact-Decorated Surfaces](#).
3. [Cation- \$\pi\$  interactions and Salt Bridges](#)
4. [Noncovalent Bond Finder](#).
5. [MSA3D: Multiple Sequence Alignment Coloring](#)
6. [Chime's Command Language](#).

Spin Zoom+ - Bkg Water Ligand 2°  
Center... Stereo Slab [PE Site Map](#)

More advanced rendering methods are available through the Advanced Explorer Menu.

## Protein Explorer Surfaces Menu

### Surfaces: 2HLA.pdb.gz ?

[Protein Explorer © 2005 by Eric Martz](#)

Calculate Surface1 for protein

coloring it (see also [Plain Colors](#))

Plain

Detail level High (water accessible)

Surface: Make Solid/Transparent/Hide Delete

[Contact Surfaces](#) [Better MEP](#)

Spin Zoom+ - Bkg Water Ligand 2°  
Center... Stereo Slab [PE Site Map](#)

#### Plain Colors

You can color a surface any arbitrary color by entering the command "color list pe\_surfN [color]", where N is the surface number in the above menu (1-8), and [color] is a [color name](#) or [RGB value](#).

#### Advanced

Spacing 0.8 Probe radius 1.4 Surface smoothing ☐

☐ Hide portion of surface within 4.1 Angstroms of

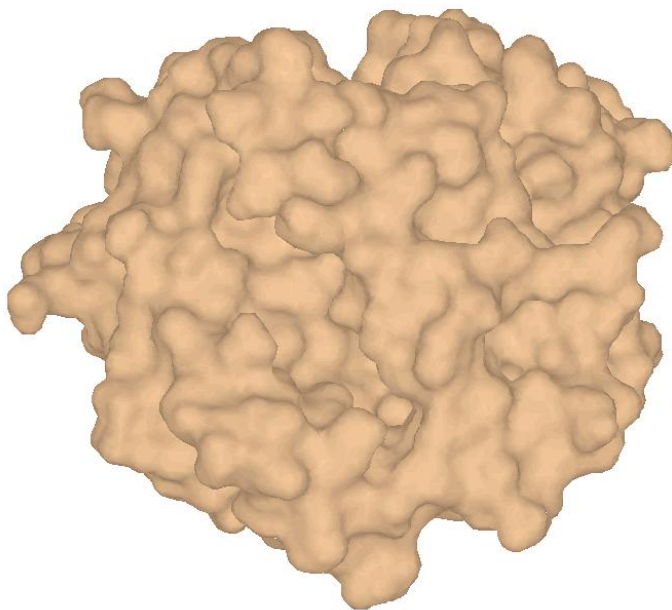
ligand

MEP function: Coulomb

The Surfaces menu allows the

user to display the surface of the protein. Several variable are available, including the radius of the probe used to define the surface, as well as several methods of coloring the surface based on chemical and physical properties.

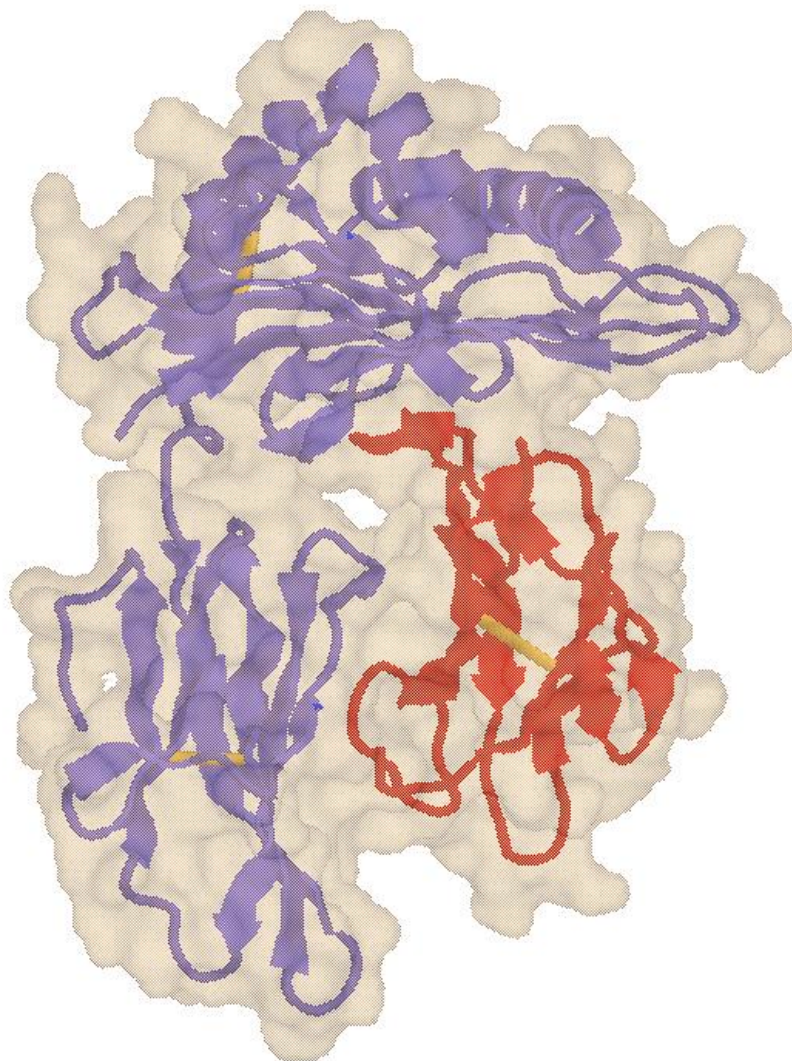
### Protein Explorer: HLA-AW Surface Rendering



MDL

This rendering style shows the surface of the protein accessible to water. This image is tilted 90 degrees toward the viewer from the previous images.

## Protein Explorer: HLA-AW Superimposed Images



MDL

By setting the surface to be transparent, it is possible to superimpose another rendering style over it, and see how it fits into the surface. This can convey an idea of how the fold of the chain relates to the overall three-dimensional shape of the protein.

## Recommended Reading and Resources:

- A detailed introduction to protein structure and function can be found in most introductory biochemistry textbooks. For example, Lehninger Principles of Biochemistry, 4th Edition, by D. L. Nelson and M. Cox (sections 2.1, 3.1-3.5, 4.1-4.4, 5.1-5.3).
- [The Structures of Life](#) at the NIH web site. This site is an introduction to protein structure, structure determination methods, drug design techniques, and other applications of structural biology.
- **Protein Structure and Function**, by Gregory A. Petsko and Dagmar Ringe. This book provides an overview of the basic biochemistry of structural biology. Topics covered include protein structure, mechanisms of protein function, regulation of protein function, and case studies of the kinds of problems that arise in structural biology.
- [The MIT Biology Hypertextbook](#). This online textbook provides introductory level coverage of the field of microbiology. It includes cell biology, protein biochemistry, genetics, metabolism, and molecular biology. New content is typically added over time.
- [Artificial Intelligence and Molecular Biology](#). This online book includes chapters on classifying protein structures, predicting protein structure, and analyzing crystallographic and NMR data to determine protein structure. Of particular interest to readers of the current page who have a computer science background but need to understand more of the basic underlying biology is [Chapter 1: Molecular Biology for Computer Scientists](#).

## Representing Proteins in Silico and Protein Forward Kinematics

### Topics in this Module

- [Modeling Proteins on a Computer](#)
  - [Cartesian Representation of Protein Conformations](#)
  - [The Internal Degrees of Freedom of a Protein](#)
  - [Dihedral Representation of Protein Conformations](#)
- [Protein Forward Kinematics](#)
  - [Mathematical Background: Matrices and Transformations](#)
  - [Forward Kinematics](#)
    - [A simple approach](#)
    - [Denavit-Hartenberg Local Frames](#)

### Modeling Proteins on a Computer

In order to construct efficient, maintainable software to deal with and manipulate protein structures, a suitable way to store these structures has to be adopted. Depending on the ultimate application, different representations may have advantages and disadvantages from a software perspective. For example, when designing a simple visualization software, the Cartesian (x,y,z) coordinates of each atom are useful and simple to render on the screen. However, if the program is to manipulate bond angles and bond lengths for example, a representation based on the internal degrees of freedom (see below) may be more appropriate. Some applications may even need to store more than one representation at a time; for example a simulation program that needs to compute a protein's Potential Energy, which is a function of both Cartesian and Internal coordinates, would benefit from keeping both representations at the same time.

The **structure** of a protein is the set of atoms it contains, and the bonds that join them, that is, its inherent connectivity. A particular geometric shape of a protein (that is, the spatial arrangement of the atoms in the molecule) is called its **conformation**. Thus, a given protein structure can have many different conformations. Next, we discuss the two most common ways to

model protein structures and conformations for software applications:  
Cartesian and Dihedral representations.

## Cartesian Representation of Protein Conformations

The most essential information for modeling a protein structure is the relative position of each atom, given as (x,y,z) Cartesian coordinates. Popular imaging methods such as X-Ray Crystallography, Nuclear Magnetic Resonance (NMR) and Cryogenic Electron Microscopy (Cryo-EM) are used to experimentally obtain relative atom positions from protein crystals and solutions. This is precisely the information provided by Protein Databank (PDB) format coordinate files:

First 19 atom coordinate records of PDB entry 2HLA

ATOM	1	N	GLY	A	1	44.842	51.034	101.284	0.01	27.20
ATOM	2	CA	GLY	A	1	45.640	50.230	100.389	0.01	26.99
ATOM	3	C	GLY	A	1	46.692	49.648	101.308	0.01	26.80
ATOM	4	O	GLY	A	1	46.895	50.222	102.381	0.01	26.91
ATOM	5	N	SER	A	2	47.283	48.516	100.951	1.00	26.26
ATOM	6	CA	SER	A	2	48.277	47.866	101.761	1.00	26.17
ATOM	7	C	SER	A	2	49.212	47.031	100.845	1.00	24.21
ATOM	8	O	SER	A	2	49.060	47.195	99.630	1.00	19.77
ATOM	9	CB	SER	A	2	47.438	47.091	102.800	1.00	26.31
ATOM	10	OG	SER	A	2	46.276	46.356	102.404	1.00	27.99
ATOM	11	N	HIS	A	3	50.147	46.186	101.370	1.00	23.93
ATOM	12	CA	HIS	A	3	51.129	45.389	100.609	1.00	21.44
ATOM	13	C	HIS	A	3	50.953	43.905	100.849	1.00	20.32
ATOM	14	O	HIS	A	3	50.530	43.595	101.950	1.00	22.00
ATOM	15	CB	HIS	A	3	52.555	45.674	100.990	1.00	19.69
ATOM	16	CG	HIS	A	3	52.940	47.090	100.611	1.00	21.44
ATOM	17	ND1	HIS	A	3	53.371	47.470	99.422	1.00	20.87
ATOM	18	CD2	HIS	A	3	52.956	48.175	101.433	1.00	21.69
ATOM	19	CE1	HIS	A	3	53.676	48.730	99.476	1.00	20.57

The third column lists the atom type and the seventh, eighth, and ninth columns contain the x, y, and z coordinates of each atom. These Cartesian coordinates are given in relation to some reference frame determined by the experimental imaging technique, which is not important. The conformation is uniquely specified by the relative positioning of the atoms.

The coordinates and type of each atom, together with the amino acid type they belong to, are sufficient information to reconstruct the connectivity (bonding) of a protein, and therefore sufficient to render an image of the protein. If one wishes to allow the protein to move in a realistic fashion, however, more information may be necessary.

## **The Internal Degrees of Freedom of a Protein**

The **degrees of freedom** of a system are a set of parameters that may be varied independently to define the state of the system. For example, the location of a point in the Cartesian 2D plane may be defined as a displacement along the x-axis and a displacement along the y-axis, given as a  $(x,y)$  pair. It may also be given as a rotation about the origin by  $\theta$  degrees and a distance  $r$  from the origin, given as a  $(r,\theta)$  pair. In either case, a point moving freely in a plane has exactly two degrees of freedom.

As mentioned before, the spatial arrangement of the atoms in a protein constitute its conformation. In the PDB coordinate file above, we can see that one obvious way to define a protein conformation is by giving  $x$ ,  $y$ , and  $z$  coordinates for each atom, relative to some arbitrary origin. These are not independent degrees of freedom, however, because atoms within a molecule are not allowed to leave the vicinity of their neighboring atoms (if no chemical reaction takes place). Pairs of atoms bonded to each other, for example, are constrained to remain close, so moving one atom causes others connected to it to move in a dependent fashion. In the kinematics terminology, this means that the true, effective or independent number of degrees of freedom is much less than the input space parameters -an  $(x,y,z)$  tuple for each atom-. The remainder of this section defines a set of independent degrees of freedom that more readily model how proteins and other organic molecules can actually move.

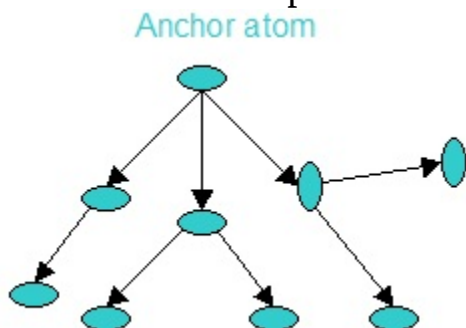
### **Bonds and Bond Length**

The atoms in proteins are connected to one another through covalent bonds. Each pair of bonded atoms has a preferred separation distance called the

**bond length.** The bond length can vary slightly with a spring-like vibration, and is thus a degree of freedom, but realistic variations in bond length are so small that most simulations assume it is fixed for any pair of atoms. This is a very common assumption in the literature and reduces the effective degrees of freedom of a protein; the remainder of this module makes this assumption.

Although bond lengths will not be allowed to vary in this work, the presence of bonds is still important because it allows us to represent the connectivity of the protein as an undirected graph data structure, where the atoms are the nodes and the bonds between them are undirected edges. In some cases, it is helpful to artificially break any cycles in the graph, and choose an atom from the interior as an anchor atom. The graph can then be treated as a tree data structure, with the anchor atom as the root.

A Protein as a Graph Data Structure



A tree-like representation of protein connectivity, for a very small molecule. Cycles are broken by ignoring one bond in each.

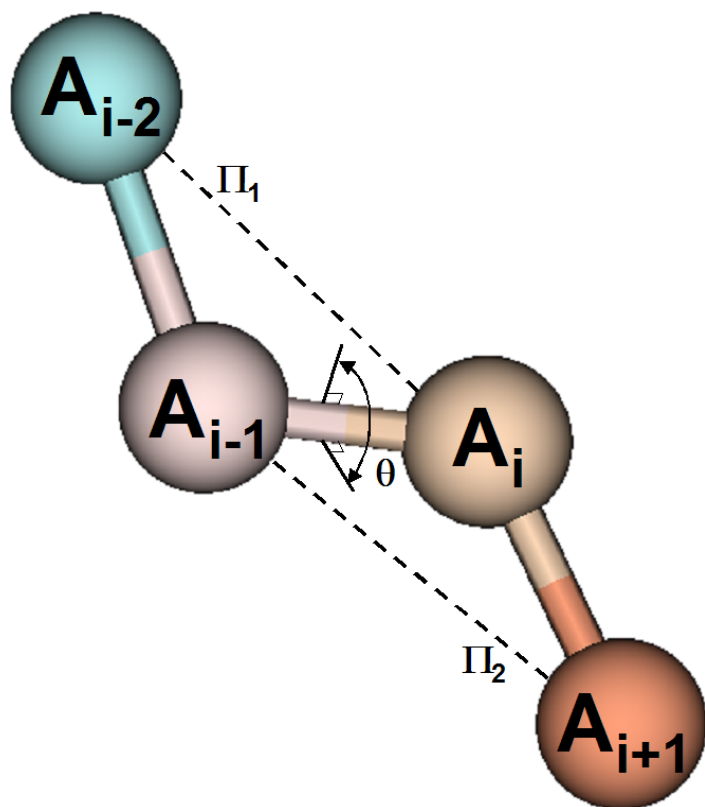
**Bond Angles**

Bond length is an independent degree of freedom given two connected atoms. A set of three atoms bonded in sequence defines another degree of freedom: the angle between the two adjacent bonds. This is, appropriately, referred to as the **bond angle**. The bond angle can be calculated as the angle between the two vectors corresponding to the bonds from the central atom to each of its neighbors. As a reminder, the angle between two vectors is the inverse cosine of the ratio of the dot product of the vectors to the product of their lengths. Like bond lengths, bond angles tend to be characteristic of the atom types involved, and, with few exceptions, vary little. Thus, like bond lengths, this module considers all bond angles as fixed (again, this is a common assumption).

### Dihedral Angles

In most organic molecules, including proteins, the most important internal degree of freedom is rotation about **dihedral (torsional) angles**. A dihedral angle is defined by four consecutively bonded atoms. Given four consecutive atoms  $A_{i-2}$ ,  $A_{i-1}$ ,  $A_i$ , and  $A_{i+1}$ , the dihedral angle is defined as the smallest angle between the planes  $\pi_1$  and  $\pi_2$ , as shown in the figure. Variation of the dihedral angle is a consequence of rotation of the two outer bonds about the central bond.

#### A Dihedral Angle



$\pi_1$  is the plane uniquely defined by the first three atoms  $A_{i-2}$ ,  $A_{i-1}$ , and  $A_i$ .

Similarly,  $\pi_2$  is the plane uniquely defined by the last three atoms  $A_{i-1}$ , and  $A_i$ , and  $A_{i+1}$ . The dihedral angle,

$\theta$ , is defined as the smallest angle between these two planes. You can read more about the [angle between two intersecting planes here](#).

In this module, because bond lengths and bond angles are being ignored as underlying degrees of freedom of a protein, the only remaining degrees of freedom are the dihedral rotations. Representing protein conformations with the dihedral angles as the only underlying degrees of freedom is known as the **idealized** or **rigid geometry model**. Ignoring bond lengths and bond angles greatly reduces the number of degrees of freedom and therefore the

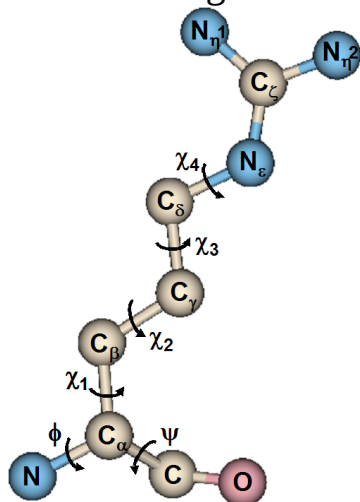
computational complexity of representing and manipulating protein structures. Even more efficient representations which reduce the number of degrees of freedom even further exist [\[link\]](#), but these are beyond the scope of this introduction.

## Dihedral Representation of Protein Conformations

All amino acids share the same core of one nitrogen, two carbon, and one oxygen atoms. This shared core makes up the backbone of the protein. There are two freely rotatable backbone dihedral angles per amino acid residue in the protein chain: the first, designated  $\varphi$ , is a consequence of the rotation about the bond between  $N$  and  $C_\alpha$ , and the other,  $\psi$ , which is a consequence of the rotation about the bond between  $C_\alpha$  and  $C$ . The peptide bond between  $C$  of one residue and  $N$  of the adjacent residue is not rotatable.

The number of backbone dihedrals per amino acid is 2, but the number of side chain dihedrals varies with the length of the side chain. Its value ranges from 0, in the case of glycine, which has no sidechain, to 5 in the case of arginine.

### Dihedral Angles in Arginine



The backbone  
atoms appear at  
the bottom of the

illustration (the peptide bond is not rotatable). The sidechain dihedrals are conventionally designated by  $\chi$  and a subscript.

One can generate different three dimensional structures of the same protein by varying the dihedral angles. There are  $2N$  backbone dihedral DOFs for a protein with  $N$  amino acids, and up to  $4N$  side chain dihedrals that one can vary to generate new protein conformations. Changes in backbone dihedral angles generally have a greater effect on the overall shape of the protein than changes in side chain dihedral angles. Think about why.

## Protein Forward Kinematics

**Kinematics** is a branch of mechanics concerned with how objects move in the absence of mass (inertia) and forces. You can imagine that varying the dihedral angles will move a protein's atoms relative to each other in space. The problem of computing the new spatial locations of the atoms given a set of dihedral rotations is known as the **forward kinematics** problem.

The importance of this problem to protein modeling and simulation should be clear: as stated earlier, the only internal degrees of freedom usually considered for a protein are its dihedral angles. Thus, moving a protein will be achieved by setting some of its dihedral angles to new values. For some applications, such as the rendering of an image of the protein and the computation of its Energy, however, the Cartesian (x,y,z) coordinates for each atom are needed. These are obtained by forward kinematics.

## Mathematical Background: Matrices and Transformations

The math involved in solving forward kinematics requires some background in linear algebra, specifically in the anatomy and application of transformation matrices. The links provided in this section should provide enough mathematical background to understand the rest of this module and eventually write a simple protein manipulation program.

## **Background on Transformations**

- **Transformation Matrices:** The main transformations you will apply to polypeptide chains will be a combination of **translations** and **rotations**. Please see [introduction to translations](#) and [introduction to two- and three-dimensional rotations](#). One special rotation matrix is the [Euler matrix](#). Please pay particular attention to the different conventions used for defining the Euler matrix. The one adopted for this module is the XYZ convention (there is also the ZXZ convention). Now that you know what an Euler matrix looks like, you need to get familiar with rotations about an arbitrary vector or line. Please read more on [rotations around an arbitrary vector](#).
- **Homogeneous Transformations:** The use of homogenous coordinates and transformations can simplify some of the calculations involved in using three-dimensional transformations. In particular, they allow **translation**, which is not a linear operator in 3D, to become a linear operator in the 3D subspace (x,y,z,1) of a 4D space. The advantage of this representation is that translation becomes achievable by multiplying a vector by a matrix, and so becomes composable. A direct benefit from this is the ability to express, as a matrix, a rotation around an **arbitrary point**, not just the origin as in the pure 3D case. See [homogenous transformations](#).
- **Quaternions** Quaternions are an efficient, robust method of representing three-dimensional rotations. In particular, they are not subject to the undesirable singularities and numerical instability of rotations represented by orthonormal matrices and Euler angles. Please visit this introduction to [quaternions](#) to see how they relate to homogenous transformations. In this class quaternions will be used for the optimal structural alignment of two proteins and it is recommended that the reader familiarizes him/herself with the concept of quaternions as soon as possible.

A more detailed discussion of spatial descriptions and transformations can be found in chapter 2 of [\[link\]](#). The most widely used transformations to manipulate protein chains are rotations. Several representations are possible for rotations:

- **Euler angles:** The orientation of an object is given as three rotations about set axes. For example, in the ZXZ convention, the angles specify a rotation about the global z-axis, followed by one about the global x-axis, and finally, one more about the global z-axis. The use of Euler angles is subject to an undesirable phenomenon called **gimbal lock**, in which two of the rotational axes become aligned in such a way that a degree of freedom is lost.
- **Cardan angles:** The orientation is specified as a set of three rotations about axes defined by the object. The typical example is the pitch-roll-yaw set of rotations for an aircraft. Pitch corresponds to a rotation about the axis from wingtip to wingtip. Roll corresponds to a rotation about an axis from the nose to the tail, and yaw corresponds to rotation about a third "vertical" axis through the center of the plane, and roughly corresponds to a notion of horizontal heading. This method is also subject to gimbal lock.
- **Axis-angle representation:** It can be proven that any three-dimensional rotation can be represented as a single rotation about an axis, represented by a unit vector.
- **Rotation matrices:** A rotation matrix is an orthonormal matrix that represents a rotation. Rotation matrices are discussed later in the module. Applying the matrix to a vector yields the rotated vector. Given two rotations represented by matrices A and B, the result of applying both rotations in sequences is given by the matrix product AB.
- **Unit quaternions:** A rotation of angle  $\theta$  about the axis represented by the unit vector  $\mathbf{v} = [x, y, z]$  is represented by a unit quaternion. Quaternions are described in [this module](#).

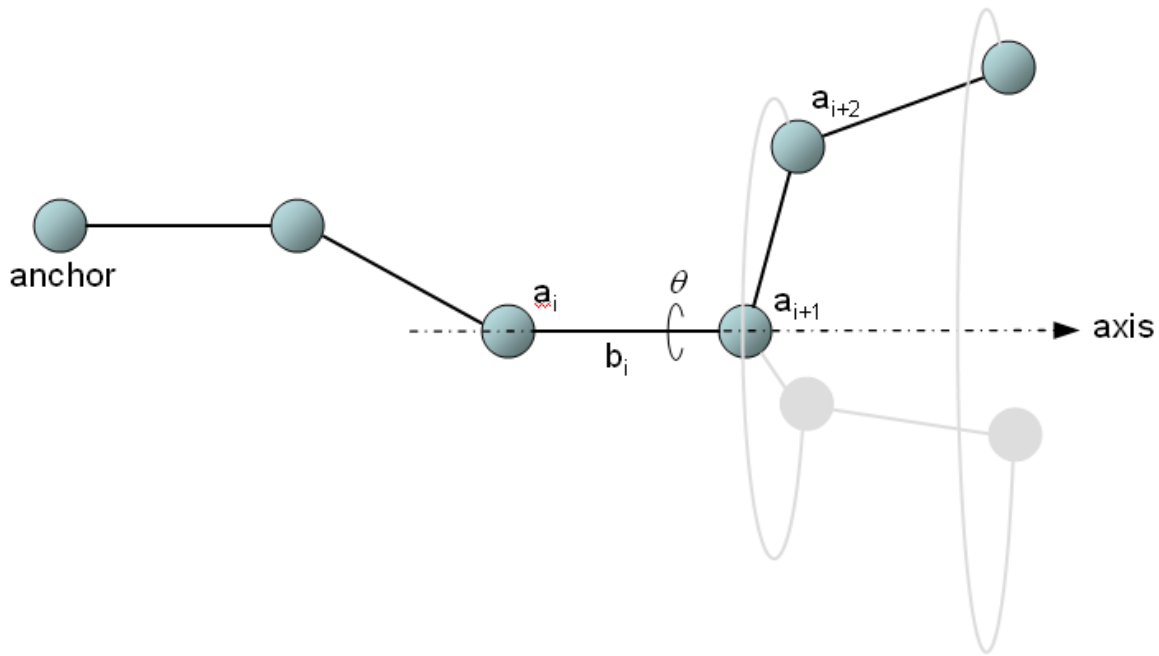
## Forward Kinematics

As stated earlier, a common operation when manipulating proteins in silico is to retrieve the Cartesian coordinates of each atom in the protein from our knowledge of its dihedral angles and rotations applied to them. For simplicity, assume we have an anchor atom and we are modeling the protein backbone only, that is, the protein consists of a serial linkage composed of consecutive backbone atoms, as shown in Figure 5.

### **A Simple Approach**

The simplest way to represent a protein chain is to store the Cartesian (x,y,z) coordinates of each atom at all times. These coordinates are relative to some global coordinate frame which is unimportant, for example that in which the atomic positions were obtained by X-Ray crystallography and which are typically read from the PDB files. These coordinates can be changed if so desired. Common changes are to remove the center of mass (thus centering the protein at the global origin), subtract the position of the anchor atom (to center the protein at this atom), etc.

But it was discussed earlier that the "natural" degrees of freedom for kinematic manipulations are usually the dihedral angles alone. This means that algorithms that operate on dihedral angles to achieve their goals will normally require a way to modify the Cartesian coordinates when dihedral rotations are performed, to reflect the new atomic positions. This can be easily done with rotation matrices as follows.



A protein backbone as a serial linkage.

When a rotation of  $\theta$  degrees around bond  $i$  is performed, one can think of all atom positions starting at  $i+2$  rotating around the axis defined by bond  $i$ , and all other atoms (from anchor to atom  $i+1$  inclusive) remaining stationary. Thus, upon such a rotation, the Cartesian coordinates of the atoms after the bond need to be updated, and their new values are given by:

$$[x', y', z', 1]^T = R(i, \theta) \cdot [x, y, z, 1]^T$$

Where  $[x, y, z, 1]$  is the position of a generic atom in homogeneous form,  $[x', y', z', 1]$  is its position after the rotation (T is the transpose operator), and  $R(i, \theta)$  is a 4x4 matrix that encodes a rotation of  $\theta$  degrees around an axis coinciding with bond  $i$  that passes through atom  $a_i$ , and is given in homogeneous form as:

$$R(i, \theta) = T(\mathbf{a}_i) \cdot R_0(\mathbf{axis}, \theta) \cdot T(-\mathbf{a}_i)$$

In the above formula,  $T(\mathbf{x})$  is a translation by the vector  $\mathbf{x}$  and  $R_0(\mathbf{axis}, \theta)$  is a rotation around an axis that goes through the origin of the specified coordinate system. As can be seen, this rotation around an arbitrary point is realized by translating the point to the origin, rotating the target atom around the axis through the origin, and then translating it back (the composition of these 3 transformations yields a unique 4x4 homogeneous matrix that achieves the same effect). The axis of rotation can easily be computed from the positions of atoms  $i$  and  $i+1$  and must have unit norm. To perform successive rotations about different bonds, this procedure can be repeated, updating the Cartesian coordinates for each rotation. Note that the convention used for matrix-vector multiplication is to multiply **column** vectors by matrices on the **left**, so the rightmost transformation gets applied first, and so on. This is the convention used in most of the literature, but the alternate convention is possible (multiplying **row** vectors with matrices on the **right**; these matrices are the transpose of the column-vector convention).

Alternatively, if many rotations need to be performed at the same time (and the intermediate Cartesian coordinates are not needed), these rotations could be sorted by bond number and applied simultaneously, by noting that rotations can be performed in a cumulative way as the backbone is traversed from anchor to end atom. The ability to chain rotations around arbitrary vectors in space (i.e. not through the origin) is one of the main benefits of homogeneous transformations. For example, if two rotations need to be applied at the same time, one around bond 3 by 30 degrees and another around bond 7 by 15 degrees, the atoms between bonds 3 and 7 get updated by:

$$[x', y', z', 1]^T = R(\mathbf{bond}_3, 30) \cdot [x, y, z, 1]^T$$

But the atoms after bond 7 are updated by:

$$[x', y', z', 1]^T = R(\mathbf{bond}_7, 15) \cdot R(\mathbf{bond}_3, 30) \cdot [x, y, z, 1]^T$$

In the above, **bond n** is the unit vector defined along bond n, easily computed by subtracting the coordinates of atoms n+1 and n, and then dividing by its norm. The chaining of transformations as explained above is very useful to achieve arbitrary rotations of bonds within a protein. Sections of the protein (i.e. atoms belonging to certain residues) can be updated when a dihedral rotation is performed simply by constructing the overall matrix that should affect them.

### Denavit-Hartenberg Local Frames

The previous approach, while simple and intuitive, has some shortcomings:

- The accumulation of math operations in the rotation matrices is prone to numerical instability. After only a couple of hundred rotations of a point, each accumulating on the other, the final position of the point may start differing significantly from its actual, intended position. As a consequence, the relative position and orientation of atoms in the protein chain will no longer be in agreement with the protein structure. In particular, bond lengths and angles will begin stretching and deviating from their physically acceptable values.
- The actual values of the Cartesian coordinates are always stored in a particular, arbitrarily chosen frame of reference. For example, if we wanted to translate the protein, we would need to modify the Cartesian coordinates stored.
- Once a rotation is applied, the method "forgets" the current values of the dihedral angles, which would need to be re-computed if needed. What is stored is a snapshot of the current Cartesian coordinates of each atom.

The original definition of Forward Kinematics, however, is a method to obtain the Cartesian coordinates of each atom from the current values of the internal degrees of freedom (dihedral angles in our case) at any time. In such an approach, the Cartesian coordinates need not be recomputed after

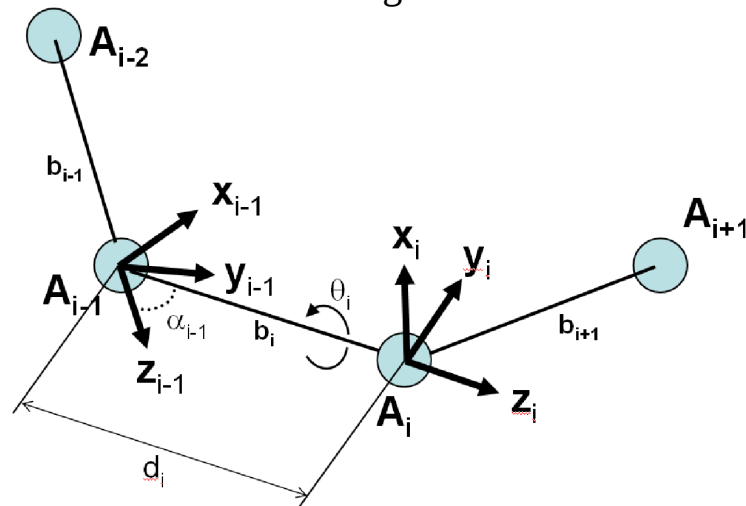
every change in the dihedral angles; rather, the idea is to store the current values of the dihedral angles, and to have a procedure to reconstruct the atomic positions when needed. The advantages of this approach are:

- A more compact representation of the variables of the problem, since the dihedral angles require less space than the (x,y,z) coordinates of each atom (the protein topology requires the values of the bond lengths and angles anyway, so the total amount of numbers to store is comparable).
- It is not prone to numerical instability since the number of rotations performed to position an atom is always its sequence number in the chain. (Actually if the chain is thousands of residues long, some uncertainty could arise in the position of atoms far along the chain, but the relative position of consecutive atoms can still be kept under control, avoiding bond stretching).
- Performing a dihedral rotation consists simply of adding/subtracting the rotation angle from the stored value for each angle. In particular, simultaneous rotations (i.e. rotating more than one dihedral angle at a time) which consists of multiplying many 4x4 matrices in the global method, reduces to modifying the angle values.
- There is no explicit global coordinate frame for the protein. It can be positioned arbitrarily by prepending a position/orientation matrix to the forward kinematics computation.

The only preprocessing step that is necessary to start working with this method, however, is to perform an initial pass on the protein to extract the initial values of the dihedral angles and the constant bond lengths and angles, from the Cartesian coordinates available from PDB files, if the intention is to start from the protein's native state. This is easily done; bond lengths can be obtained by computing the distance between the bonded atoms, and bond angles by computing the angle between the vectors formed by two consecutive bonds (recall that the dot product of two vectors yields the product of their lengths times the cosine of the angle between them). Next, we present the transformations required for the Denavit-Hartenberg method.

Consider three consecutive bonds as in the figure below. Suppose that a local coordinate frame is attached at the beginning of each bond. For example, local coordinate system  $x_{i-1}, y_{i-1}, z_{i-1}$  is centered at atom  $A_{i-1}$ . Therefore, imagine that the position of each atom in three-dimensional space is specified in terms of a frame that is centered at the previous atom. Given the frames at atom  $A_{i-2}$ , and atom  $A_{i-1}$ , one can determine how the frames at atoms  $A_i$  and atom  $A_{i+1}$  will change in space as a consequence of a rotation around the bond that connects atoms  $A_{i-1}$  and  $A_i$  with the dihedral angle [\[link\]](#). The correct transformation can be computed in terms of three primitive operations: two rotations and one translation. The two rotations are a rotation around the dihedral bond by the dihedral angle and a rotation around an axis perpendicular to the bond angle, by the bond angle. The translation refers to the fact that the origins of the frames are on the respective centers of the atoms connected by the bond, thus separated by bond lengths.

The Denavit-Hartenberg convention



To describe the position of atom  $i$  in terms of the coordinate frame centered at atom  $i-1$ , two rotations and a translation are composed.

The order in which to compose these 3 transformations, to obtain the total transformation that expresses the position of atom  $i$  in terms of frame  $i-1$ , is

the following:

$$R(\mathbf{x}, \alpha_{i-1}) \cdot R(\mathbf{z}, \theta_i) \cdot T(0, 0, d_i)$$

where the rotation axes are the usual x (1,0,0) and z (0,0,1), not to be confused with the DH Local Frames. The resulting homogenous transformation is shown below.

Transformation

$$T = \begin{bmatrix} \cos(\theta_i) & -\sin(\theta_i) & 0 & 0 \\ \sin(\theta_i) \cos(\alpha_{i-1}) & \cos(\theta_i) \cos(\alpha_{i-1}) & -\sin(\alpha_{i-1}) & -\sin(\alpha_{i-1})d_i \\ \sin(\theta_i) \sin(\alpha_{i-1}) & \cos(\theta_i) \sin(\alpha_{i-1}) & \cos(\alpha_{i-1}) & \cos(\alpha_{i-1})d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Homogeneous transformation to express the coordinates of atom  $i$  in terms of the frame centered at  $i-1$

Note that  $\theta_i$  is the dihedral angle on bond  $b_i$  and  $\alpha_{i-1}$  is the bond angle between bonds  $b_{i-1}$  and  $b_i$ .  $d_i$  is the length of bond  $b_i$ . For a more detailed derivation of this transformation, please read the included material in required readings. The position of any atom in the molecule can be determined by chaining matrices of the form given above. For example, suppose that  $b_i, b_{i-1}, \dots, b_1$ , represents the sequence of bonds on the path from a particular atom  $a$  to the anchor atom  $a_{\text{anch}}$ . Then, for atom  $a$ , its Cartesian coordinates with respect to the frame attached to the anchor atom is given by:

Equation 1

$$\begin{bmatrix} x_i & y_i & z_i & 1 \end{bmatrix}^t = T_1 T_2 \dots T_i \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix}^t$$

The coordinates of atom  $a$  with respect to the local frame attached to it are 0, 0, 0.

To complete the description, one can allow for rotations or translations of the local frame attached to the anchor atom with respect to some global frame. Rotations of the anchor atom with respect to a global frame cause a rigid rotation of the entire polypeptide chain. To do so, one can define the rotation frame as the Euler matrix defined by the Euler angles of the local frame of the anchor atom to the global frame. As discussed before, there are many conventions to define the Euler matrix. One of them, the X-Y-Z convention, defines the Euler matrix as the product of three rotation matrices: rotation around the z axis by angle  $\alpha$ ; rotation around the y axis by angle  $\beta$ ; rotation around the x axis by the angle  $\gamma$ . The order of performing these three rotations in the X-Y-Z conventions is: rotation around x axis first, then around y axis second, and around z axis last. The resulting Euler matrix according to this convention is given below:

Euler Matrix

$$E = \begin{pmatrix} c\alpha c\beta & c\alpha s\beta s\gamma - s\alpha c\gamma & c\alpha s\beta c\gamma + s\alpha s\gamma & 0 \\ s\alpha c\beta & s\alpha s\beta s\gamma + c\alpha c\gamma & s\alpha s\beta c\gamma - c\alpha s\gamma & 0 \\ -s\beta & c\beta s\gamma & c\beta c\gamma & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$\alpha$ ,  $\beta$ ,  $\gamma$  are the so-called Euler angles - the angles with respect to each of the Cartesian axes. The convention used here is the XYZ convention.  $c\alpha$  and  $s\alpha$  denote  $\cos(\alpha)$  and  $\sin(\alpha)$  respectively.

The Euler matrix can be applied last to the accumulating dihedral rotations in order to allow the anchor atom to move with respect to a global frame. For a more detailed explanation, please read the included material in required readings.

## Required Reading

- [Zhang-Kavraki 2002 \[PDF\]](#). Zhang, M. and L. E. Kavraki, "A New Method for Fast and Accurate Derivation of Molecular Conformations". Journal of Chemical Information and Computer Sciences, 42:64-70, 2002.
- Stamati-Shehu-Kavraki. Computing Forward Kinematics for Protein-like linear systems using Denavit-Hartenberg Local Frames [\[PDF\]](#)

## Resources

1. **VMD** Visual Molecular Dynamics, is an excellent tool for visualization and scripted manipulation of protein structures that uses Tcl scripting - Humphrey, W., Dalke, A. and Schulten, K., "VMD - Visual Molecular Dynamics", J. Molec. Graphics, 1996, vol. 14, pp. 33-38.
2. **RasMol** is mostly a viewer, but has some built-in tools. - Roger Sayle and E. James Milner-White. "RasMol: Biomolecular graphics for all", Trends in Biochemical Sciences (TIBS), September 1995, Vol. 20, No. 9, p. 374.
3. **Chimera** is a very powerful visualizer that handles huge structures easily. - Pettersen, E.F., Goddard, T.D., Huang, C.C., Couch, G.S., Greenblatt, D.M., Meng, E.C., and Ferrin, T.E. "UCSF Chimera - A Visualization System for Exploratory Research and Analysis." J. Comput. Chem. 25(13):1605-1612 (2004).
4. **InsightII**, **Cerius2** and **Catalyst** are products for simulation, discovery and analysis that recently became commercial, and can be found [here](#).
5. **CHARMM** is a simulation package based on the CHARMM force field. Brooks BR, Bruccoleri RE, Olafson BD, States DJ, Swaminathan S, Karplus M (1983). "CHARMM: A program for macromolecular energy, minimization, and dynamics calculations". J Comp Chem 4: 187217.
6. **NAMD** is another popular simulation package and can be obtained [here](#). James C. Phillips, Rosemary Braun, Wei Wang, James Gumbart, Emad Tajkhorshid, Elizabeth Villa, Christophe Chipot, Robert D. Skeel, Laxmikant Kale, and Klaus Schulten. Scalable molecular dynamics with NAMD. Journal of Computational Chemistry, 26:1781-1802, 2005.

7. **Amber** is one of the most widely used molecular dynamics simulators due to its speed. Duan et al. A point-charge force field for molecular mechanics simulations of proteins based on condensed-phase quantum mechanical calculations Journal of Computational Chemistry Vol. 24, Issue 16. Pages 1999-2012 (2003).

## Protein Inverse Kinematics and the Loop Closure Problem

- [Background Material](#)
- [Inverse Kinematics and its Relevance to Proteins](#)
- [Solving Inverse Kinematics](#)
  - [Inverse Kinematics Methods](#)
    - [Classical Methods](#)
    - [Optimization-Based Methods](#)
  - [Cyclic Coordinate Descent and Its Application to Proteins](#)

## Background Material

The math involved in solving the Inverse Kinematics problem requires some background in linear algebra, specifically in the anatomy and application of transformation matrices. Please refer to [Forward Kinematics](#) for an introduction to transformation matrices. It is very important that you understand how to apply transformations for the Forward Kinematics of a chain.

## Inverse Kinematics and its Relevance to Proteins

**Inverse kinematics** (IK) is the problem of finding the right values for the underlying degrees of freedom of a chain, in the case of a protein polypeptide chain, of the dihedral angles, so that the chain satisfies certain spatial constraints. For example, in some applications, it is necessary to find rotations that can steer certain atoms to desired locations in space. To achieve a particular function, protein regions sometimes have to undergo concerted motion where atoms move together in order to locate themselves near another protein or molecule. The motion of atoms is spatially constrained because they have to assume specific target locations in space. However, since atoms must move together in order not to break bonds by their motion, it is easier to model their motion in **dihedral angle space**, where bond lengths and bond angles are fixed. This parameterization of

protein motion, called the **idealized** or **rigid geometry model**, is discussed in [Representing Proteins in silico: Data Structures and Kinematics](#).

Solving the Inverse Kinematics problem in the context of proteins, i.e., finding what values of the dihedral angles of a protein polypeptide chain yield configurations of the chain where the endpoints satisfy spatial constraints, is a very important problem in structural biology. The relevance of Inverse Kinematics for proteins can be seen in three main applications:

- Finding a missing loop (Loop Closure Problem)
- Characterizing the Flexibility of a fragment of the protein polypeptide chain
- Generating ensembles of protein structures

It is worth noting that many globular proteins have a relatively stable, inflexible core region consisting of tightly arranged secondary structure elements. However, proteins are less compact and more flexible at the surface, where unstructured fragments of the protein polypeptide chain, mobile loops, may swing freely. One consequence of loop mobility is that experimental structure determination methods may have difficulty resolving the atomic positions of surface loops. The positions of the atoms in mobile loops may be so inconsistent that no single position relative to the core dominates. In such cases, experimental structure determination methods cannot determine the positions of the atoms of a mobile loop.

When this happens, the result is a partially resolved protein structure, with fragments of the protein chain, such as mobile loops, missing. The only information available for the missing fragment is its amino acid sequence and where its two endpoints need to be spatially located in order to connect with the known, resolved, part of the protein structure. Given the spatial constraints on the endpoints of the missing fragment, one needs to find values to the dihedral angles of the fragment in order to obtain configurations of the fragment consistent with the constraints. This problem is known as the Loop Closure problem in the structural biology community. It is easy to note that even though this problem is cast in the context of finding atomic positions of a missing fragment such as a mobile loop, it is nothing new but a statement of the Inverse Kinematics problem for proteins.

Solving the Inverse Kinematics problem in the context of a missing fragment in proteins is not limited to finding mobile loops. More generally, through the Inverse Kinematics problem, one can search for alternative configurations of any fragment of a protein polypeptide chain (even fragments containing secondary structural elements) that satisfy the spatial constraints on their endpoints. Very recently, a third application has emerged, where alternative configurations of consecutive fragments that cover a polypeptide chain are generated to obtain an ensemble of alternative protein structures.

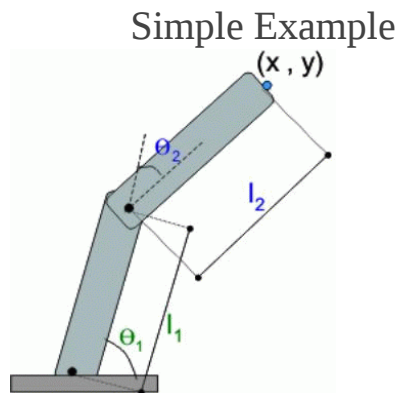
## Solving Inverse Kinematics

In applying inverse kinematics algorithms to proteins, we are taking advantage of a striking similarity between organic molecules and robotic manipulators (robot arms) in terms of how they move. As robot manipulators have joints, proteins have atoms. As robot manipulators have links that connect their joints, proteins have bonds that connect their atoms. The similarity between proteins and robots makes it possible for us to apply to proteins a large existing literature of solutions to the Inverse Kinematics problem, developed in the context of robot **manipulators** (robotic arms).

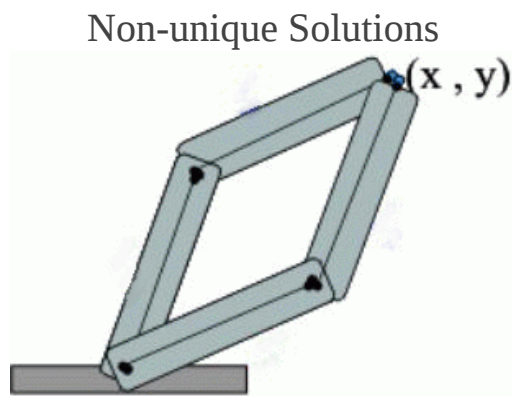
Before we proceed with some simple inverse kinematics examples, note that inverse kinematics is the inverse of the forward kinematics problem. Therefore, an immediate attempt to solve the inverse kinematics problem would be by inverting forward kinematics equations.

Let's illustrate how to solve the inverse kinematics problem for robot manipulators on a simple example. The figure below shows a simple planar robot with two arms. The underlying degrees of freedom of this robot are the two angles dictating the rotation of the arms. These are labeled in the figure below as  $\theta_1$  and  $\theta_2$ . The inverse kinematics question in this case would be: What are the values for the degrees of freedom so that the end effector of this robot (the tip of the last arm) lies at position (x,y) in the two-dimensional Cartesian space? One straightforward approach to solving the problem is to try to write down the forward kinematics equations that relate (x,y) to the two rotational degrees of freedom (see [Forward Kinematics](#) for

details on how to do so), then try to solve these equations. The solutions will give you an answer to the inverse kinematics problem for this robot.



Steer end-effector to  $(x, y)$  target position.



Two solutions depicted for this IK problem.

Illustration of solving the Inverse Kinematics

problem for a simple  
planar robot with two  
arms. Figure is adapted  
from [MathWorks](#).

### Exercise:

#### Problem:

Given an (x, y) target position for the end-effector of a robot with only two degrees of freedom  $\theta_1$  and  $\theta_2$ , what are the solutions for  $\theta_1$  and  $\theta_2$ ?

---

#### Solution:

You can compare your answer with the derivation steps below.

#### Simple Example Solved

$$x = l_1 \cos(\theta_1) + l_2 \cos(\theta_1 + \theta_2)$$

$$y = l_1 \sin(\theta_1) + l_2 \sin(\theta_1 + \theta_2)$$

$$x^2 + y^2 = l_1^2 + l_2^2 + 2l_1 l_2 \cos(\theta_2)$$

$$\cos(\theta_2) = \frac{x^2 + y^2 - l_1^2 - l_2^2}{2l_1 l_2}$$

$$x = l_1 \cos(\theta_1) + l_2 (\cos(\theta_1) \cos(\theta_2) - \sin(\theta_1) \sin(\theta_2))$$

$$x = \cos(\theta_1) (l_1 + l_2 \cos(\theta_2)) - \sin(\theta_1) (l_2 \sin(\theta_2))$$

$$y = \cos(\theta_1) (l_2 \sin(\theta_2)) + \sin(\theta_1) (l_1 + l_2 \cos(\theta_2))$$

$$\cos(\theta_1) = \frac{x + \sin(\theta_1) l_2 \sin(\theta_2)}{l_1 + l_2 \cos(\theta_2)}$$

$$\sin(\theta_1) = \frac{(l_1 + l_2 \cos(\theta_2))y - l_2 \sin(\theta_2)x}{l_1^2 + l_2^2 + 2l_1 l_2 \cos(\theta_2)}$$

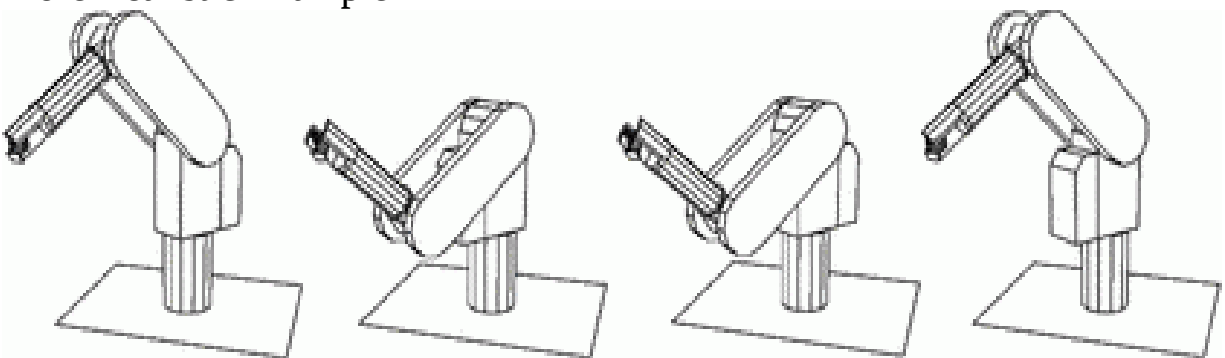
Finding solutions to  $\theta_1$  and  $\theta_2$  from the forward kinematics equations of the 2-arm planar robot.

You can see that there can be 0, 1, or 2 solutions for this example. Where does the non-uniqueness of the solutions lie in the answers we derive?

As it can be seen in the example above, the solutions to an inverse kinematics problem are not necessarily unique. In fact, as the number of degrees of freedom increases, so does the maximum number of solutions, as depicted in the figure. It is also possible for a problem to have no solution if the point on the robot cannot be brought to the target point in space at all.

While the above example offers equations that are easy to solve, general inverse kinematics problems require solving systems of nonlinear equations for which there are no general algorithms. Some inverse kinematics problems cannot be solved analytically. In robotics, it is sometimes possible to design systems to have solvable inverse kinematics, but in the general case, we must rely on approximation methods in order to keep the problem tractable, or, in some cases, even solvable. For examples on how to address inverse kinematics in particular robotic systems, please read chapter 4 of [\[link\]](#). An illustration of the solutions of the inverse kinematics problem for a robot which is widely used in industry is shown below.

**More Realistic Example**



The spatial constraint on the end-effector of this three-dimensional

manipulator can be satisfied by a maximum of four different configurations of the manipulator. Figure is obtained from [Serial Robots](#).

## Inverse Kinematics Methods

Inverse kinematics methods are categorized into two main groups:

- exact, classic, or algebraic methods
- heuristic, or optimization methods

While exact methods are **complete**, i.e. they report all solutions, they can only find solutions for chains with up to nine degrees of freedom. Hierarchical approaches break long chains into smaller ones for which exact methods provide answers. More powerful methods, referred to as optimization or heuristic methods, though not complete, are unrestricted in the number of degrees of freedom in the systems about which they reason.

### Classic Inverse Kinematics Methods

It is known that for manipulators with no more than six degrees of freedom, there is a finite number of solutions to the inverse kinematics problem [\[link\]](#). There is, however, no analytical method that can find these solutions for all types of manipulators. For manipulators with only revolute joints, which is the case for biomolecules with idealized geometry, the number of unique solutions is at most 16, when the number of degrees of freedom does not exceed six [\[link\]](#). An efficient solution was proposed in [\[link\]](#) and later applied to the conformational analysis of small molecular chains [\[link\]](#), [\[link\]](#). Methods based on curve approximation were proposed in [\[link\]](#) for the inverse kinematics of hyper-redundant robots, where the number of regularly distributed joints is very large.

Specialized solutions to inverse kinematics in biochemistry appeared as early as 1970 [\[link\]](#), where fragments of up to 6 degrees of freedom were predicted by solving a set of polynomial equations representing geometric transformations. These equations were applied to building tripeptide loops [\[link\]](#) under the ideal geometry assumption. Later work [\[link\]](#), [\[link\]](#), [\[link\]](#), [\[link\]](#) offered efficient analytical solutions for three consecutive residues through spherical geometry and polynomial equations. Bounding inverse kinematic solutions for chains with no more than six degrees of freedom within small intervals has also been shown relevant in the context of drug design [\[link\]](#). A new formulation that extends the domain of solutions to any three residues, not necessarily consecutive, and with arbitrary geometry, was recently proposed [\[link\]](#). Current work that pushes the dimensionality limit from six to nine degrees of freedom makes use of an efficient subdivision of the solution space [\[link\]](#).

#### **Inverse Kinematics Methods with Optimization**

Currently, optimization-based solutions are considered most appropriate for accommodating chains with arbitrary numbers of degrees of freedom. Two well-known optimization-based inverse kinematics solutions that iteratively solve a system of equations until loops are closed are **Random Tweak** [\[link\]](#), [\[link\]](#) and **Cyclic Coordinate Descent (CCD)** [\[link\]](#), [\[link\]](#), [\[link\]](#). Both methods are based on iteratively setting the dihedral degrees of freedom of a fragment or kinematic chain until the end effector (atom for a protein) reaches a target position.

Random tweak relies on the computation of the Jacobian (a high-dimensional analog of the derivative of a function on real numbers), a process that is computationally expensive and numerically unstable. In addition to not being free from mathematical singularities, random tweak does not allow additional constraints on individual residues because modifications to dihedral angles are introduced all at once, with a strong dependence of each dihedral proposed change on all the others. Additional constraints on the dihedrals may result in the unpredictable motion of a feature atom away from rather than toward its target position.

Avoiding the use of the Jacobian, CCD is computationally inexpensive, numerically stable, and free from singularities. CCD avoids the interdependence of dihedral angles by adjusting only a single degree of freedom at a time. This allows for additional constraints on dihedral angles with a predictable motion of the end effector towards the target position. First introduced in the context of non-linear programming [\[link\]](#), CCD was found applications in the robotics [\[link\]](#) community, and later in the structural biology community in the context of the loop closure problem for proteins[\[link\]](#).

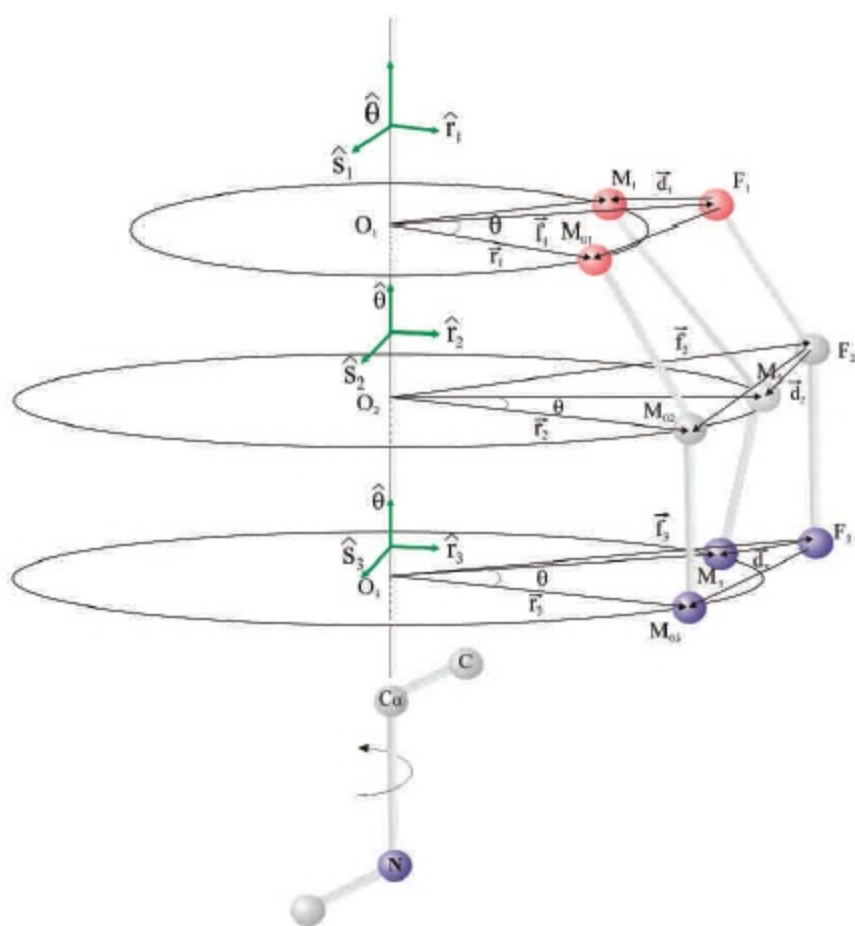
## Cyclic Coordinate Descent and Its Application to Proteins

CCD tries to find an optimal angle by which to rotate a single bond so as to steer a desired atom towards its target position. When applying CCD to find dihedral angles of a fragment of the polypeptide chain so that the ends of the fragment connect properly with the rest of the chain, it is important to steer not just one atom of the end of the fragment, but the three backbone atoms of the end simultaneously. Finding values to the dihedral angles that steer the three backbone atoms of the end of the fragment simultaneously to their target positions guarantees that the end of the fragment will assume both its target position and orientation in space. We will explain how to find optimal values to the dihedral angles of a fragment by which to simultaneously steer the three backbone atoms of the end of the fragment to their target positions. We first define their current positions  $M$  and their target positions  $F$ , as shown in the figure below. The goal is to minimize the Euclidean distance between the current and the target positions for all three atoms simultaneously.

In order to find the optimal angle by which to rotate a particular bond, we can define an objective function  $S$  that we wish to minimize. We propose a value for  $S$  that sums the square of the deviations between the final position of the atoms after the rotation ( $M$ ) and the desired positions ( $F$ ). Using this nomenclature, the squared norm of the vector  $M-F$  (denoted  $FM$ ) has precisely this value for each of the three atoms, so we can sum the three contributions to  $S$ . The  $FM$  vectors can be defined relative to an origin  $O$  located along the axis of rotation, which will simplify the math, since the

rotation is two-dimensional when working on the plane perpendicular to the axis of rotation.  $O$  can be computed by projecting the current position of the atom,  $M$ , onto the rotation axis. It is convenient to decompose  $OM$  for each feature atom into two components (along the  $r$  and  $s$  local axes), in order to allow its expression in terms of the angle being rotated (using cosine and sine). This way, the distance between the atoms and their target positions will be only a function of the fixed (rotatable bond) atoms and the angle to rotate, which remains the only variable and the problem can be solved.

### CCD Schematics



Find optimal dihedral rotation for the current bond  
so that all three desired atoms reach their target  
positions.

## Finding Optimal Angle

$$S = |\vec{F}_1 M_1|^2 + |\vec{F}_2 M_2|^2 + |\vec{F}_3 M_3|^2$$

$$\vec{F}_1 M_1 = \vec{O}_1 M_1 - \vec{O}_1 F_1$$

$$\vec{O}_1 M_1 = r_1 \cos \theta \hat{r}_1 + r_1 \sin \theta \hat{s}_1$$

$$\vec{F}_1 M_1 = r_1 \cos \theta \hat{r}_1 + r_1 \sin \theta \hat{s}_1 - \vec{f}_1 = \vec{d}_1$$

$$|\vec{d}_1|^2 = r_1^2 + f_1^2 - 2r_1 \cos \theta (\vec{f}_1 \cdot \hat{r}_1) - 2r_1 \sin \theta (\vec{f}_1 \cdot \hat{s}_1)$$

$$|\vec{d}_2|^2 = r_2^2 + f_2^2 - 2r_2 \cos \theta (\vec{f}_2 \cdot \hat{r}_2) - 2r_2 \sin \theta (\vec{f}_2 \cdot \hat{s}_2)$$

$$|\vec{d}_3|^2 = r_3^2 + f_3^2 - 2r_3 \cos \theta (\vec{f}_3 \cdot \hat{r}_3) - 2r_3 \sin \theta (\vec{f}_3 \cdot \hat{s}_3)$$

Since S is defined as the sum of squared distances between current positions and target positions, steering these three atoms to their target positions requires minimizing S. Therefore, the optimal dihedral rotation can be found by minimizing S.

## Schematics of Cyclic Coordinate Descent

The question then becomes that of finding a rotation along the rotation axis O, shown in the figure, that minimizes S. First, we need to define S as a function of the angle we are trying to find. Doing so is not hard, since rotation by this angle is a two-dimensional rotation. In the figure above we have shown how the position of an atom can be updated through a two-dimensional rotation by the angle around the rotation axis. In this way we obtain an expression that relates S to the angle we want to find. Since this angle has to minimize S, it has to provide a solution to the first order

derivative of S set to 0. This is shown below in Figure 5. Simplifying the expression for the first order derivative of S set to 0 gives us a formula for  $\tan(\alpha)$ . CCD is a very efficient method due to the fact that it obtains the value for  $\alpha$  analytically. However, an expression for the tangent does not provide a unique value for the angle. This is a consequence of the fact that the derivative of S set to 0 corresponds not only to minima, but also to local maxima. In order to find the angle that indeed minimizes S, one would have to make sure that the second order derivative of S is greater than 0. This is more cumbersome. There is a way to avoid doing such calculations by realizing that the formula we received for S in terms of the angle we want to solve for, can be rewritten as shown in Figure 5. In this way, one can obtain a value for both the cosine and the sine of the angle, which now uniquely determines the optimal angle.

CCD Solution

$$\frac{dS}{d\theta} = \frac{d\left(\left|\vec{d}_1\right|\right)}{d\theta} + \frac{d\left(\left|\vec{d}_2\right|\right)}{d\theta} + \frac{d\left(\left|\vec{d}_3\right|\right)}{d\theta}$$

$$\frac{d\left(\left|\vec{d}_i\right|\right)}{d\theta} = 2r_i \sin\theta (\vec{J}_i \cdot \hat{r}_i) - 2r_i \cos\theta (\vec{J}_i \cdot \hat{s}_i)$$

$$\tan \alpha = \frac{(\vec{J}_1 \cdot \hat{s}_1)r_1 + (\vec{J}_2 \cdot \hat{s}_2)r_2 + (\vec{J}_3 \cdot \hat{s}_3)r_3}{(\vec{J}_1 \cdot \hat{r}_1)r_1 + (\vec{J}_2 \cdot \hat{r}_2)r_2 + (\vec{J}_3 \cdot \hat{r}_3)r_3}$$

Posing a  
minimization  
procedure reveals  
the value for the  
optimal angle.

Better Solution

$$S = a - b \cos\theta - c \sin\theta$$

$$\cos\alpha = \frac{b}{\sqrt{b^2 + c^2}}, \quad \sin\alpha = \frac{c}{\sqrt{b^2 + c^2}}$$

$$S = a - \sqrt{b^2 + c^2} \cos(\theta - \alpha)$$

The correct quadrant  
can be determined

by rewriting  $S$ .

### Solution to the minimization of $S$

Unlike classic inverse-kinematics solutions that use Jacobian matrices [\[link\]](#), [\[link\]](#), or general numerical approaches, CCD is free of singularities and does not depend on initial guesses for solutions. Compared to inverse kinematics techniques with optimization that suffer from high computational times, CCD is computationally fast. Unlike other methods such as random tweak, CCD gives predictable behavior and suffers from no anomalies when additional constraints are added to the dihedrals (e.g. constraints imposed by the physical-chemical forces on the protein). Such properties make CCD very appealing.

Because CCD solves for the degrees of freedom of a chain one at a time, the method finds the optimal values for all the degrees of freedom of the chain iteratively, according to some order. CCD iterates over the degrees of freedom according to a predetermined order (e.g. a straightforward implementation of the method may use the identity order, where degrees of freedom are numbered consecutively in increasing order from the base to the end effector of the chain), solving for each one of them at a time. This process of iterating over all the degrees of freedom can be repeated a maximum number of times or until the end effector lies within an epsilon distance of its position and orientation in space.

While not able to enumerate all the solutions to the degrees of freedom, CCD guarantees it will find a solution if one exists. Given a configuration of the chain and a target position and orientation for the chain's end effector, CCD iteratively modifies the degrees of freedom of the chain until either it runs out of iterations or it manages to satisfy the spatial constraint on the end effector. Due to its computational efficiency (linear time complexity on the number of degrees of freedom of the chain), CCD has been applied to

determine atomic positions of missing mobile loops of arbitrary length[\[link\]](#). A similar application complete missing loops in partially resolved crystallographic structures can be found in [\[link\]](#), [\[link\]](#), [\[link\]](#).

A recent application of CCD to not just loops but any fragment of a protein polypeptide chain can be found in[\[link\]](#). The work in [\[link\]](#) applies CCD to configurations of a fragment that are sampled uniformly at random to obtain an ensemble of fragment configurations that connects with the rest of the protein polypeptide chain. Careful attention is paid to the energetic refinement of the obtained fragment configurations in order to ensure the biological relevance of the configurations at room temperature. The usage of CCD in [\[link\]](#) to obtain an ensemble of biologically meaningful configurations of a fragment of the polypeptide chain is an interesting application to capture the flexibility of a fragment in the context of a given protein structure. By generating ensembles of biologically relevant configurations for fragments that are defined consecutively and with significant overlap over the protein polypeptide chain, the work in [\[link\]](#) offers a novel approach to capture the flexibility of the entire polypeptide chain.

## Recommended Reading

- Canutescu, A.A. and R.L. Dunbrack. [\[PDF\]](#). Cyclic Coordinate Descent: A Robotics Algorithm for Protein Loop Closure. Protein Science, 12:963-72, 2003.
- Craig, J.J. Introduction to Robotics, chapter 4. Reading, MA: Addison-Wesley, 1989.
- van den Bedem, H. and Lotan, I. and Deacon, A. M. and Latombe, J.-C.. [\[PDF\]](#). Computing protein structures from electron density maps: the missing loop problem. Algorithmic Foundations of Robotics VI, 345-360, 2005.
- Shehu, A. and Clementi, C. and Kavraki, L.E. [\[PDF\]](#). Modeling Protein Conformational Ensembles: From Missing Loops to Equilibrium Fluctuations. Proteins: Structure, Function, Bioinformatics, 65(1):164-179, 2006.

- Coutsiyas, E. A. and Seok, C. and Wester, M. J. and Dill, K. A. [\[LINK\]](#). Resultants and loop closure. International Journal of Quantum Chemistry, 106(1):176-189,2005.

## Molecular Shapes and Surfaces

### Topics in this Module

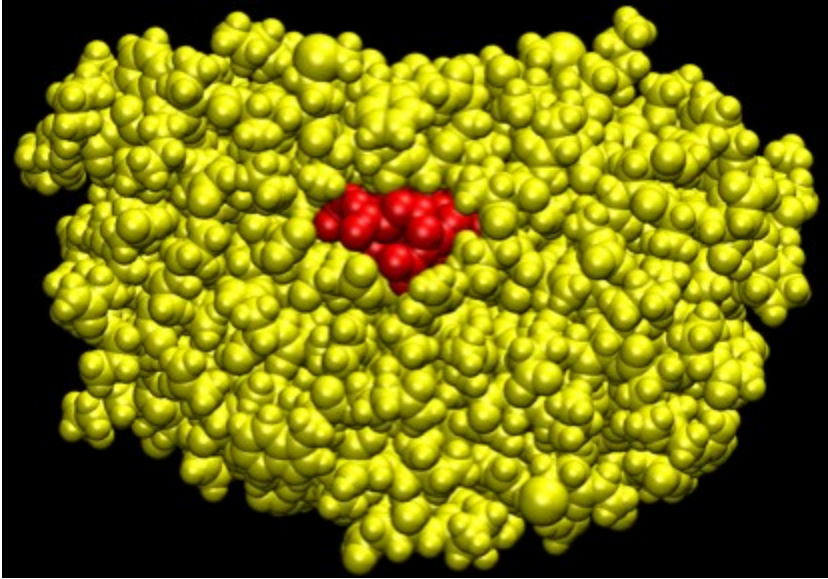
- [Introduction](#)
- [Representing Shape](#)
- [Alpha-Shapes](#)
  - [Delaunay Triangulation](#)
  - [Weighted Alpha-Shapes](#)
- [Calculating Molecular Volume Using Alpha-Shapes](#)
- [Related Software](#)

### Introduction

Many problems in structural biology, require a researcher to understand the shape of a protein. At first glance, this may seem obvious. By opening a molecular visualizer, one can easily see the shape of a protein. But what about calculating the surface area or volume of the protein? What about performing analyses of the surface, such as looking for concave pockets in a protein that might be binding sites for other molecules? What about calculating the volume and shape of those empty binding pockets, in order to find molecules that might fit in them? What about determining whether a particular small molecule can fit in a binding pocket?

All of these problems require some formal notion of the shape of a protein. A protein structure file usually provides no more information than a list of atom locations in space and their types. It will be assumed that for any given application, a radius may be defined for each atom type. This leads to the space filling representation of a protein, in which each atom is treated as an impenetrable sphere.

HIV-1 Protease



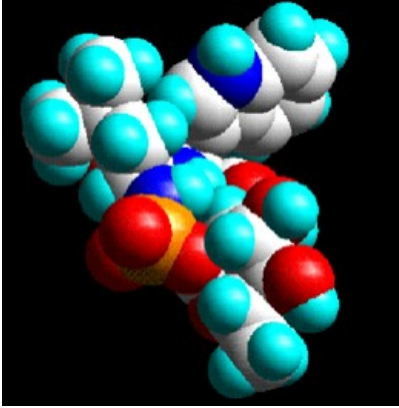
A space filling representation of HIV-1 protease (yellow) with an inhibitory drug (red) blocking its binding site.

This representation allows for visualization, but it brings us no closer to being able to computationally decide which parts of which atoms are on the surface of the protein and which are buried inside the structure. Some additional tool is needed to capture notions of interior and exterior and spatial adjacency.

## Representing Shape

Using the sphere model for atoms, one way to define the shape of a molecule is as the union of (possibly overlapping) balls in  $\mathbb{R}^3$ .

Space Filling Diagram

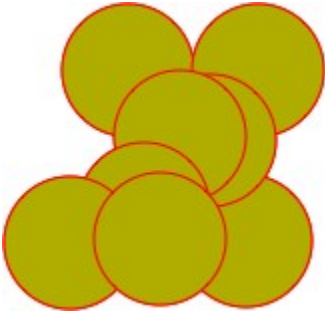


The space filling diagram models each atom as a sphere in 3D.

Since proteins inside our cells are in an aqueous environment, considering a protein's interactions with solvent molecules, particularly water, is very important for appropriately modeling them. Recall that one of the phenomena that determines the structure of a protein is the hydrophobic effect: some amino acid residues are stabilized by the presence of water, and others are repelled. The extent of the interaction of a protein with the surrounding water depends on the surface area of the protein that can be reached by water molecules. Therefore, quantitative modeling of the strength of interaction with solvent often involves computing the **solvent accessible surface area (SASA)**. Computing SASA can be done by regarding each solvent molecule as a sphere of set radius. This is of course a simplification, since water molecules are not spherical. When this sphere rolls about the molecule, its center delineates the SASA. One can think of the SASA of a molecule as the result of growing each atom sphere by the radius of the solvent sphere. Instead, by taking what is swept out by the front of the solvent sphere, we obtain the **molecular surface (MS)** model of the molecule. Alternatively, the MS can be obtained by removing a layer of solvent radius depth from the SASA model.

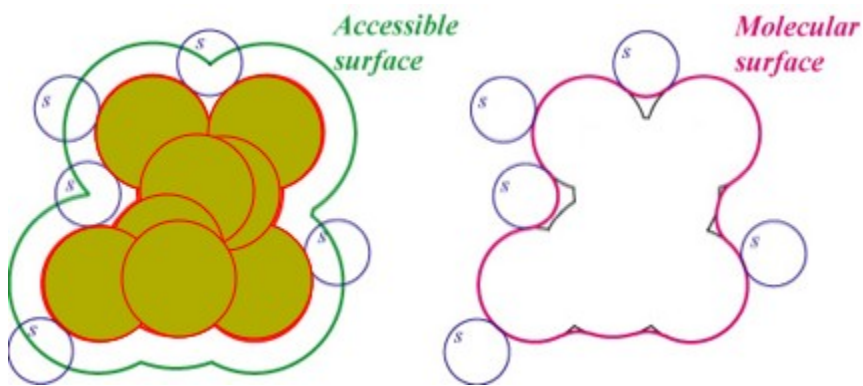
Representations of Molecular Shape

VDW Representation



Each atom can  
be modeled as  
a Van der  
Waals sphere  
in three  
dimensions.  
The union of  
the spheres  
gives the  
molecular  
surface.

### Accessible Surface Area



Not all molecular surface is accessible to solvent due to the existence of small cavities. Rolling a solvent ball over the Van der Waals spheres traces out the surface area experienced by the solvent. Solvent accessible surface area (SASA) is a very important measure for

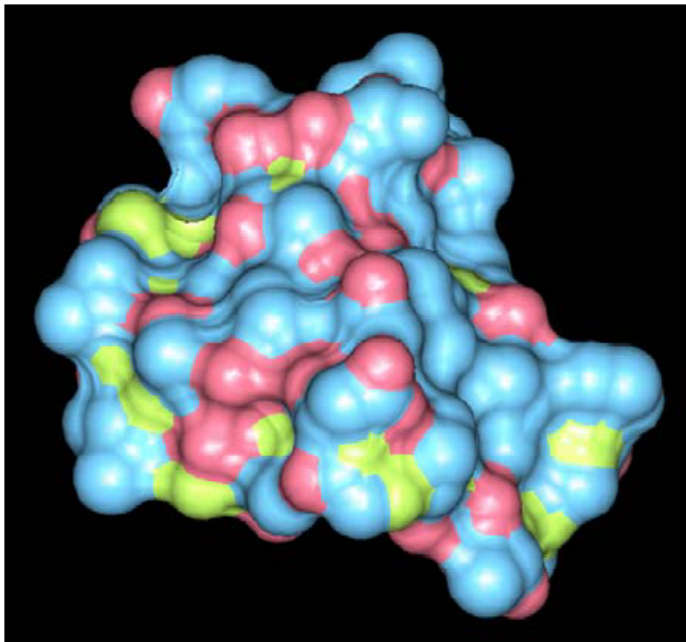
quantitatively determining the behavior and interaction tendencies of a protein.

Two different notions and representations of the surface of a molecule.

The surface determined by SASA analysis depends on the size of a typical solvent molecule. The larger the solvent, the less contoured the resulting surface will appear, because a larger probe molecules would not be able to fit into some of the interatomic spaces that a smaller one would.

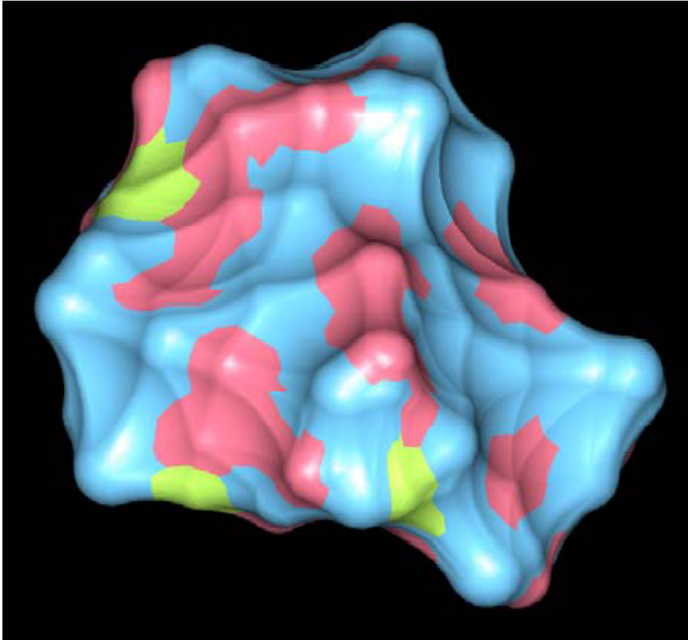
#### Solvent Accessible Surface Area

Probing the surface area with a solvent ball of radius 1.4 Å



Typically, solvent is modeled as a ball of radius 1.4 Å. This delineates the solvent accessible surface shown.

Probing the surface area with a solvent ball of radius 1.5 Å



Increasing the radius of the solvent ball reduces the solvent accessible surface area because there are more cavities that a bulkier ball cannot penetrate.

Solvent-accessible surface area (SASA) for two different solvent radii.

## Alpha-Shapes

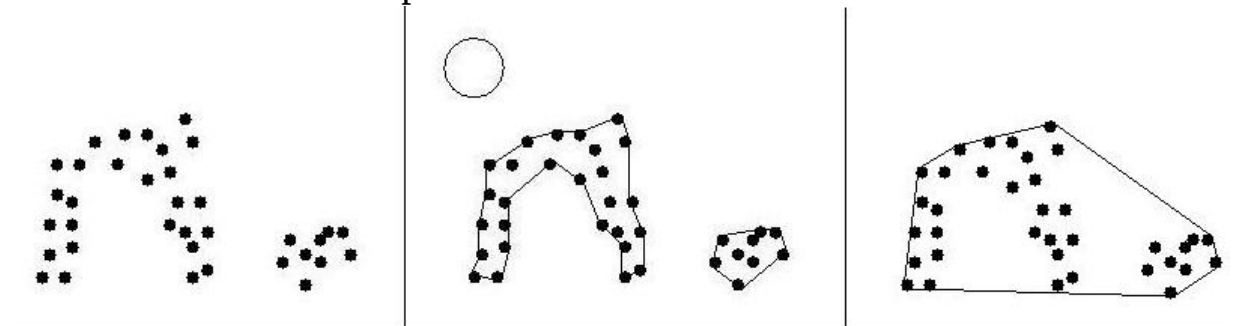
Part of the problem with defining the shape of a protein is that we start with nothing but a point set, and the "shape" of a set of discontinuous points is poorly defined. The problem is, what do we mean by shape? As you saw above, the shape of a molecule depends on what is being used to measure it. To handle this ambiguity, we will introduce a method of shape calculation based on a parameter,  $\alpha$ , which will determine the radius of a spherical

probe that will define the surface. The method defines a class of shapes, called  **$\alpha$ -shapes** [\[link\]](#) for any given point set. It allows fast, accurate, and efficient calculations of volume and surface area.

$\alpha$ -shapes are a generalization of the **convex hull**. Consider a point set  $S$ . Define an  $\alpha$ -ball as a sphere of radius  $\alpha$ . An  $\alpha$ -ball is empty if it contains no points in  $S$ . For any  $\alpha$  between zero and infinity, the  **$\alpha$ -hull** of  $S$  is the complement of the union of all empty  $\alpha$ -balls.

- For  $\alpha$  of infinity, the  $\alpha$ -shape is the convex hull of  $S$ .
- For  $\alpha$  smaller than the  $1/2$  smallest distance between two points in  $S$ , the  $\alpha$ -shape is  $S$  itself.
- For any  $\alpha$  in between, one can think of the  $\alpha$ -hull as the largest polygon (polyhedron) or set thereof whose vertices are in the point set and whose edges are of length less than  $2\alpha$ . The presence of an edge indicates that a probe of radius  $\alpha$  cannot pass between the edge endpoints.

### Two-Dimensional $\alpha$ -Shapes



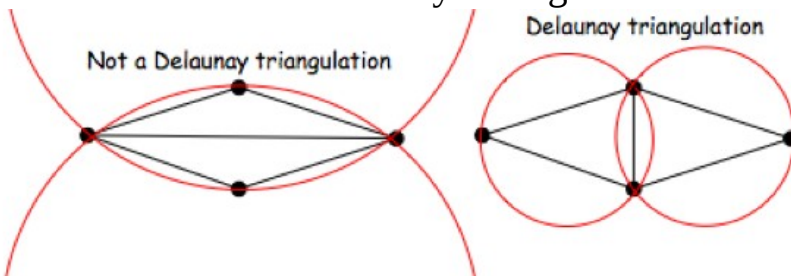
Some  $\alpha$ -shapes are shown for a point set and various values of  $\alpha$ . On the left,  $\alpha$  is 0 or slightly more, such that an  $\alpha$ -ball can fit between any two points in the set. The  $\alpha$ -shape is therefore the original point set. On the right,  $\alpha$  is infinity, so an  $\alpha$ -ball can be approximated locally by a line.  $\alpha$  on this scale yields the convex hull of the point set. The middle image shows the  $\alpha$ -shape for  $\alpha$  equal to the radius of the ball shown.

This yields two disjoint boundaries, one of which has a significant indentation. **Voids**, or empty pockets completely enclosed by the  $\alpha$ -shape, are also possible, for instance if the  $\alpha$ -shape is ring-like (in 2D) or forms a hollow shell (in 3D).

## Computing the Alpha-Shape: Delaunay Triangulation

A **triangulation** of a three-dimensional point set  $S$  is any decomposition of  $S$  into non-intersecting tetrahedra (triangles for two-dimensional point sets). The **Delaunay triangulation** of  $S$  is the unique triangulation of  $S$  satisfying the additional requirement that no sphere circumscribing a tetrahedron in the triangulation contains any point in  $S$ . Although it is incidental to  $\alpha$ -shapes, it is worth noting that the Delaunay triangulation maximizes the average of the smallest angle over all triangles. In other words, it favors relatively even-sided triangles over sharp and stretched ones.

### Two-Dimensional Delaunay Triangulation



The Delaunay triangulation of the four points given is shown on the right. Note that the circumscribing circles on the left each contain one point of  $S$ , whereas the circles on the right do not. The transition from the triangulation on the left to that on the right is called an edge flip, and is the basic operation of constructing a two-dimensional Delaunay triangulation. Face flipping is the analogous procedure for five points in three dimensions.

The Delaunay triangulation of a point set is usually calculated by an incremental flip algorithm as follows:

- The points of  $S$  are sorted on one coordinate ( $x$ ,  $y$ , or  $z$ ). This step is not strictly necessary but makes the algorithm run faster than if the points were in arbitrary order.
- Each point is added in sorted order. Upon adding a point:
  - The point is connected to previously added points that are "visible" to it, that is, to points to which it can be connected by a line segment without passing through a face of a tetrahedron.
  - Any new tetrahedra formed are checked and flipped if necessary.
  - Any tetrahedra adjacent to flipped tetrahedra are checked and flipped. This continues until further flipping is unnecessary, which is guaranteed to occur

This algorithm runs in worst case  $O(n^2)$  time, but expected  $O(n^{3/2})$  time. Without the sort in the first step, the expected case would be  $O(n \log n)$ . A full description and analysis of Delaunay triangulation algorithms is given in [1], chapter 9.

From the Delaunay triangulation the  $\alpha$ -shape is computed by removing all edges, triangles, and tetrahedra that have circumscribing spheres with radius greater than  $\alpha$ . Formally, the  **$\alpha$ -complex** is the part of the Delaunay triangulation that remains after removing edges longer than  $\alpha$ . The  **$\alpha$ -shape** is the boundary of the  $\alpha$ -complex.

Pockets [\[link\]](#) can be detected by comparing the  $\alpha$ -shape to the whole Delaunay triangulation. Missing tetrahedra represent indentations, concavity, and generally negative space in the overall volume occupied by the protein. Particularly large or deep pockets may indicate a substrate binding site.

## Weighted Alpha Shapes

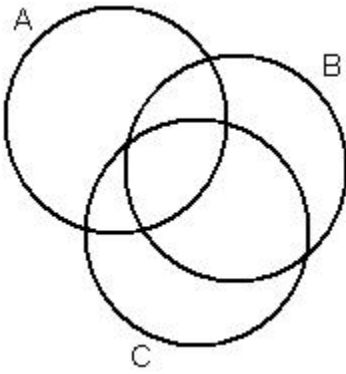
Regular  $\alpha$ -shapes can be extended to deal with varying weights (i.e., spheres with different radii, such as different types of atoms) [\[link\]](#). The formal definitions become complicated, but the key idea is to use a pseudo distance measure that uses the weights. Suppose we have two atoms at positions  $p_1$  and  $p_2$  with weights  $w_1$  and  $w_2$ . Then the pseudo distance is defined as the square of the Euclidean distance minus the weights. The pseudo distance is zero if and only if two spheres centered at  $p_1$  and  $p_2$  with radii equal to  $\sqrt{w_1}$  and  $\sqrt{w_2}$  are just touching.

$$d((p_1, w_1), (p_2, w_2)) = \|p_1 - p_2\|^2 - w_1 - w_2,$$

Pseudo distance to account  
for atoms of different sizes.

## Calculating Molecular Volume Using $\alpha$ -Shapes

The volume of a molecule can be approximated using the space-filling model, in which each atom is modeled as a ball whose radius is  $\alpha$ , where  $\alpha$  is selected depending on the model being used: Van der Waals surface, molecular surface, solvent accessible surface, etc. Unfortunately, calculating the volume is not as simple as taking the sum of the ball volumes because they may overlap. Calculating the volume of a complex of overlapping balls is non-trivial because of the overlaps. If two spheres overlap, the volume is the sum of the volumes of the spheres minus the volume of the overlap, which was counted twice. If three overlap, the volume is the sum of the ball volumes, minus the volume of each pairwise overlap, plus the volume of the three-way overlap, which was subtracted one too many times in accounting for the pairwise overlaps. In the general case, all pairwise, three-way, four-way and so on to  $n$ -way intersections (assuming there are  $n$  atoms) must be considered. Proteins generally have thousands or tens of thousands of atoms, so the general  $n$ -way case may be computationally expensive and may introduce numerical error.



Three overlapping discs (balls if three dimensional). Calculating the total area (volume if three dimensional) of the balls requires summing the areas of each ball, then subtracting out the pairwise intersection areas, since each was counted once for each ball it is inside.

Then the intersection area of the three balls must be added back because, although it was added three

times initially, it  
was also  
subtracted once  
in each of the  
three pairwise  
intersections. In  
the general case,  
with  $n$  balls, all  
of which may  
overlap,  
intersections of  
odd numbers of  
balls are added,  
and  
intersections of  
even numbers of  
balls subtracted,  
to calculate the  
total area or  
volume.

$\alpha$ -shapes provide a way around this undesirable combinatorial complexity [\[link\]](#), and this issue has been one of the motivating factors for introducing  $\alpha$ -shapes. To calculate the volume of a protein, we take the sum of all ball volumes, then subtract only those pairwise intersections for which a corresponding edge exists in the  $\alpha$ -complex. Only those three-way intersections for which the corresponding triangle is in the  $\alpha$ -complex must then be added back. Finally, only four-way intersections corresponding to tetrahedra in the  $\alpha$ -complex need to be subtracted. No higher-order intersections are necessary, and the number of volume calculations necessary corresponds directly to the complexity of the  $\alpha$ -complex, which is  $O(n \log n)$  in the number of atoms.

An example of how this approach works is given on page 4 of the [Liang et al. article](#) in the Recommended Reading section below. A proof of

correctness and derivation is also provided in the article. Surface area calculations, such as solvent-accessible surface area, which is often used to estimate the strength of interactions between a protein and the solvent molecules surrounding it, are made by a similar use of the  $\alpha$ -complex.

## Recommended Reading

- H. Edelsbrunner, D. Kirkpatrick, and R. Seidel. [\[PDF\]](#). "On the Shape of a Set of Points in the Plane." IEEE Transactions on Information Theory, 29(4):551-559, 1983. This is the original  $\alpha$ -shapes paper (caution: the definition of  $\alpha$  is different from that used in later papers-- it is the negative reciprocal of  $\alpha$  as presented above).
- H. Edelsbrunner and E.P. Mücke. [\[PDF\]](#). "Three-dimensional Alpha Shapes." Workshop on Volume Visualization, Boston, MA. pp 75-82. 1992. This article shows how to extend  $\alpha$ -shapes to three-dimensional point sets.
- J. Liang, H. Edelsbrunner, P. Fu, P.V. Sudhakar, and S. Subramaniam. [\[PDF\]](#). Analytical shape computation of macromolecules: I. molecular area and volume through alpha shape. Proteins: Structure, Function, and Genetics, 33:1-17, 1998. This is a paper on using  $\alpha$ -shapes to speed up volume and surface area calculations for molecular models.
- H. Edelsbrunner, M.Facello and Jie Liang. [\[PDF\]](#). On the definition and the construction of pockets in macromolecules. Discrete and Applied Mathematics, 88:83-102, 1998.

## Software

- [An  \$\alpha\$ -shapes applet](#). This applet lets you display  $\alpha$ -shapes, Voronoi diagrams, and Delaunay triangulations for arbitrary point sets and variable  $\alpha$  (use the slider at the bottom). Be cautioned that this applet uses the original definition of  $\alpha$ , which is  $-1/\alpha$  as we defined  $\alpha$  above for three-dimensional point sets.
- [Alpha shape software](#)
- [Delaunay triangulation applet](#)
- [A small example showing how  \$\alpha\$ -shapes can be used to identify pockets.](#)

- [Computational geometry software \(CGAL\)](#), which includes demo programs for alpha shapes, Delaunay triangulations, etc.
- [Locating binding sites in protein structures \(including use of  \$\alpha\$ -shapes\)](#).

## Molecular Distance Measures

### Topics in this Module

- [Comparing Molecular Conformations](#)
- [RMSD and IRMSD](#)
- [Optimal Alignment for IRMSD Using Rotation Matrices](#)
- [Optimal Alignment for IRMSD Using Quaternions](#)
  - [Introduction to Quaternions](#)
  - [Quaternions and Three-Dimensional Rotations](#)
  - [Optimal Alignment with Quaternions](#)
- [Intramolecular Distance and Related Measures](#)

## Comparing Molecular Conformations

Molecules are not rigid. On the contrary, they are highly flexible objects, capable of changing shape dramatically through the rotation of dihedral angles. We need a measure to express how much a molecule changes going from one conformation to another, or alternatively, how different two conformations are from each other. Each distinct shape of a given molecule is called a **conformation**. Although one could conceivably compute the volume of the intersection of the alpha shapes for two conformations (see [Molecular Shapes and Surfaces](#) for an explanation of alpha shapes) to measure the shape change, this is prohibitively computationally expensive. Simpler measures of distance between conformations have been defined, based on variables such as the Cartesian coordinates for each atom, or the bond and torsion angles within the molecule. When working with Cartesian coordinates, one can represent a molecular conformation as a vector whose components are the Cartesian coordinates of the molecule's atoms. Therefore, a conformation for a molecule with N atoms can be represented as a 3N-dimensional vector of real numbers.

## RMSD and IRMSD

One of the most widely accepted difference measures for conformations of a molecule is **least root mean square deviation (IRMSD)**. To calculate the

RMSD of a pair of structures (say  $x$  and  $y$ ), each structure must be represented as a  $3N$ -length (assuming  $N$  atoms) vector of coordinates. The RMSD is the square root of the average of the squared distances between corresponding atoms of  $x$  and  $y$ . It is a measure of the average atomic displacement between the two conformations:

$$\sqrt{\frac{1}{N} \sum_{i=1}^N |x_i - y_i|^2}$$

However, when molecular conformations are sampled from molecular dynamics or other forms of sampling, it is often the case that the molecule drifts away from the origin and rotates in an arbitrary way. The lRMSD distance aims at compensating for these facts by representing the minimum RMSD over all possible relative positions and orientations of the two conformations under consideration. Calculating the lRMSD consists of first finding an optimal alignment of the two structures, and then calculating their RMSD. Note that aligning two conformations may require both a translation and rotation. In other words, before computing the RMSD distance, it is necessary to remove the translation of the centroid of both conformations and to perform an "optimal alignment" or "optimal rotation" of them, since these two factors artificially increase the RMSD distance between them.

Finding the optimal rotation to minimize the RMSD between two point sets is a well-studied problem, and several algorithms exist. The **Kabsch Algorithm** [1][2], which is implemented in several molecular modeling packages, solves a matrix equation for the three dimensional rotation matrix corresponding to the optimal rotation. An alternative approach, discussed in detail after the matrix method, uses a compact representation of rotational transformations called **quaternions** [3][4]. Quaternions are currently the preferred representation for global rotation in calculating lRMSD, since they require less numbers to be stored and are easy to re-normalize. In contrast, re-normalization of orthonormal matrices is quite expensive and potentially numerically unstable. Both quaternions and their application to global alignment of conformations will be presented after the next section.

## Optimal Alignment for IRMSD Using Rotation Matrices

This section presents a method for computing the optimal rotation between 2 datasets as an orthonormal rotation matrix. As stated earlier, this approach is slightly more numerically unstable (since guaranteeing the orthonormality of a matrix is harder than the unit length of a quaternion) and requires taking care of the special case when the resulting matrix may not be a proper rotation, as discussed below.

As stated earlier, the optimal alignment requires both a translation and a rotation. The translational part of the alignment is easy to calculate. It can be proven that the optimal alignment is obtained by translating one set so that its centroid coincides with the other set's centroid (see section 2-C of [3] for proof). The centroid of a point set  $a$  is simply the average position of all its points:

Centroid of a Point Set

$$a_c = \frac{1}{n} \sum_{i=1}^n a_i$$

The centroid  
of a point set  
is the average  
position over  
all the points.

We can then redefine each point in two sets  $A$  and  $B$  as a deviation from the centroid:

Redefining Point Sets in Terms of Centroids

$$a'_i = a_i - a_c$$

$$b'_i = b_i - b_c$$

Each point is  
now  
expressed as  
a deviation  
from its set's  
centroid.

Given this notation relative to the centroid, we can explicitly set the centroids to be equal and proceed with the rotational part of the alignment.

One of the first references to the solution of this problem in matrix form is from Kabsch [1][2]. The Kabsch method uses [Lagrange multipliers](#) to solve a minimization problem to find the optimal rotation. Here, we present a slightly more intuitive method based on matrix algebra and properties, that achieves the same result. This formulation can be found in [4] and [5].

Imagine we wish to align two conformations composed of  $N$  atoms each, whose Cartesian coordinates are given by the vectors  $x$  and  $y$ . The main idea behind this approach is to find a 3x3 orthonormal matrix  $U$  such that the application of  $U$  to the atom positions of one of the data vectors,  $x$ , aligns it as best as possible with the other data vector,  $y$ , in the sense that the quantity to minimize is the distance  $d(Ux, y)$ , where  $x$  and  $y$  are assumed to be **centered**, that is, both their centroids coincide at the origin (centering both conformations is the first step). Mathematically, this problem can be stated as the **minimization** of the following quantity:

$$E = \frac{1}{N} \sum_{i=1}^N |Ux_i - y_i|^2$$

When  $E$  is a minimum, the square root of its value becomes the least RMSD (lRMSD) between  $x$  and  $y$ . Being an orthonormal rotation matrix,  $U$  needs to satisfy the orthonormality property  $UU^T = I$ , where  $I$  is the identity matrix. The orthonormality constraint ensures that the rows and columns are mutually orthogonal, and that their length (as vectors) is one. Any orthonormal matrix represents a rigid orientation (transformation) in space.

The only problem with this approach as is, is that all orthonormal matrices encode a rigid transformation, but if the rows/columns of the matrix do not constitute a **right handed system**, then the rotation is said to be **improper**. In an improper rotation, one of the three directions may be "mirrored". Fortunately, this case can be detected easily by computing the determinant of the matrix  $U$ , and if it is negative, correcting the matrix. Denoting  $Ux$  as  $x'$ , and moving the constant factor  $N$  to the left, the formula for the error becomes:

$$NE = \sum_{i=1}^N |x'_i - y_i|^2$$

An alternative way to represent the two point sets, rather than a one-dimensional vector or as separate atom coordinates, is using two  $3 \times N$  matrices ( $N$  atoms, 3 coordinates for each). Using this scheme,  $x$  is represented by the matrix  $X$  and  $y$  is represented by the matrix  $Y$ . Note that column  $1 \leq i \leq N$  in these matrices stands for point (atom)  $x_i$  and  $y_i$ , respectively. Using this new representation, we can write:

$$NE = \sum_{i=1}^N |x'_i - y_i|^2 = \text{Tr}((X' - Y)^T (X' - Y))$$

where  $X' = UX$  and  $\text{Tr}(A)$  stands for the [trace](#) of matrix  $A$ , the sum of its diagonal elements. It is easy to see that that the trace of the matrix to the right amounts precisely to the sum on the left (simply carrying out the multiplication of the first row/column should convince the reader). The right-hand side of the equation can be expanded into:

$$\text{Tr}((X' - Y)^T (X' - Y)) = \text{Tr}(X'^T X') + \text{Tr}(Y^T Y) - 2\text{Tr}(Y^T X')$$

Which follows from the properties of the trace operator, namely:  $\text{Tr}(A+B)=\text{Tr}(A)+\text{Tr}(B)$ ,  $\text{Tr}(AB)=\text{Tr}(BA)$ ,  $\text{Tr}(A^T)=\text{Tr}(A)$ , and  $\text{Tr}(kA)=k\text{Tr}(A)$ . Furthermore, the first two terms in the expansion above represent the sum of the squares of the components  $x_i$  and  $y_i$ , so it can be rewritten as:

$$NE = \sum_{i=1}^N (|x_i|^2 + |y_i|^2) - 2\text{Tr}(Y^T X')$$

Note that the  $x$  components do not need to be primed (i.e.,  $x'$ ) since the rotation  $U$  around the origin does not change the length of  $x_i$ . Note that the summation above does not depend on  $U$ , so **minimizing**  $E$  is equivalent to **maximizing**  $\text{Tr}(Y^T X')$ . For this reason, the rest of the discussion focuses on finding a proper rotation matrix  $U$  that maximizes  $\text{Tr}(Y^T X')$ .

Remembering that  $X' = UX$ , the quantity to maximize is then  $\text{Tr}(Y^T U X)$ . From the property of the trace operator, this is equivalent to  $\text{Tr}((XY^T)U)$ . Since  $XY^T$  is a square  $3 \times 3$  matrix, it can be decomposed through the [Singular Value Decomposition](#) technique (SVD) into  $XY^T = VSW^T$ , where  $V$  and  $W^T$  are the matrices of left and right eigenvectors (which are orthonormal matrices), respectively, and  $S$  is a diagonal  $3 \times 3$  matrix containing the eigenvalues  $s_1, s_2, s_3$  in decreasing order. Again from the properties of the trace operator, we obtain that:

$$\text{Tr}(Y^T X') = \text{Tr}(VSW^T U) = \text{Tr}(SW^T UV) = \sum_{i=1}^3 s_i w_i^T U v_i$$

If we introduce the  $3 \times 3$  matrix  $T$  as the product  $T = W^T UV$ , we can rewrite the above expression as:

$$\text{Tr}(Y^T X') = \sum_{i=1}^3 s_i T_{ii} \leq \sum_{i=1}^3 s_i$$

Since  $T$  is the product of orthonormal matrices, it is itself an orthonormal matrix and  $\det(T) = +1$ . This means that the absolute value of each element of this matrix is no more than one, from where the last equality follows. It is obvious that the maximum value of the left hand side of the equation is reached when the diagonal elements of  $T$  are equal to 1, and since it is an orthonormal matrix, all other elements must be zero. This results in  $T = I$ . Moreover, since  $T = W^T U V$ , we can write that  $W^T U V = I$ , and because  $W$  and  $V$  are orthonormal,  $W W^T = I$  and  $V V^T = I$ . Multiplying  $W^T U V$  by  $W$  to the left and  $V^T$  to the right yields a solution for  $U$ :

$$U = W V^T$$

Where  $V$  and  $W^T$  are the matrices of left and right eigenvectors, respectively, of the covariance matrix  $C = X Y^T$ . This formula ensures that  $U$  is orthonormal (the reader should carry out the high-level matrix multiplication and verify this fact).

The only remaining detail to take care of is to make sure that  $U$  is a **proper** rotation, as discussed before. It could indeed happen that  $\det(U) = -1$  if its rows/columns do not make up a right-handed system. When this happens, we need to compromise between two goals: maximizing  $\text{Tr}(Y^T X')$  and respecting the constraint that  $\det(U) = +1$ . Therefore, we need to settle for the second largest value of  $\text{Tr}(Y^T X')$ . It is easy to see what the second largest value is; since:

$$\text{Tr}(Y^T X') = \sum_{i=1}^3 s_i T_{ii} = s_1 T_{11} + s_2 T_{22} + s_3 T_{33}$$

$$\text{where } s_1 \geq s_2 \geq s_3 \geq 0 \text{ and } |T_{ii}| \leq 1$$

then the second largest value occurs when  $T_{11} = T_{22} = +1$  and  $T_{33} = -1$ . Now, we have that  $T$  cannot be the identity matrix as before, but instead it

has the lower-right corner set to -1. Now we finally have a unified way to represent the solution. If  $\det(C) > 0$ ,  $T$  is the identity; otherwise, it has a -1 as its last element. Finally, these facts can be expressed in a single formula for the optimal rotation  $U$  by stating:

$$U = W \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & d \end{pmatrix} V^T$$

where  $d = \text{sign}(\det(C))$ . In the light of the preceding derivation, all the facts that have been presented as a proof can be succinctly put as an algorithm for computing the optimal rotation to align two data sets  $x$  and  $y$ :

### **Optimal rotation**

1. Build the 3xN matrices  $X$  and  $Y$  containing, for the sets  $x$  and  $y$  respectively, the coordinates for each of the N atoms after centering the atoms by subtracting the centroids.
2. Compute the covariance matrix  $C = XY^T$
3. Compute the SVD (Singular Value Decomposition) of  $C = VSW^T$
4. Compute  $d = \text{sign}(\det(C))$
5. Compute the optimal rotation  $U$  as

$$U = W \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & d \end{pmatrix} V^T$$

### **Optimal Alignment for IRMSD Using Quaternions**

Another way of solving the optimal rotation for the purposes of computing the IRMSD between two conformations is to use **quaternions**. These provide a very compact way of representing rotations (only 4 numbers as compared to 9 or 16 for a rotation matrix) and are extremely easy to normalize after performing operations on them. Next, a general introduction

to quaternions is given, and then they will be used to compute the optimal rotation between two point sets.

## Introduction to Quaternions

Quaternions are an extension of complex numbers. Recall that complex numbers are numbers of the form  $a + bi$ , where  $a$  and  $b$  are real numbers and  $i$  is the canonical imaginary number, equal to the square root of  $-1$ . Quaternions add two more imaginary numbers,  $j$  and  $k$ . These numbers are related by the set of equalities in the following figure:

Equation Relating the Imaginary Elements  $i$ ,  $j$  and  $k$

$$i^2 = j^2 = k^2 = i \cdot j \cdot k = -1$$

Properties of quaternion arithmetic follow directly from these equalities.

These equalities give rise to some unusual properties, especially with respect to multiplication.

Multiplication Table for the Imaginary Elements  $i$ ,  $j$  and  $k$

$$i \cdot j = k \qquad j \cdot i = -k$$

$$j \cdot k = i \qquad k \cdot j = -i$$

$$k \cdot i = j \qquad i \cdot k = -j$$

Note that multiplication of  $i$ ,  $j$ , and  $k$  is **anti-commutative**.

Given this definition of i, j, and k, we can now define a quaternion.

#### Definition of a Quaternion

$$a + b \cdot i + c \cdot j + d \cdot k$$

A quaternion is a number of the above form, where a, b, c, and d are real-valued scalars and i, j, and k are imaginary numbers as defined above.

Based on the definitions of i, j and k, we can also derive rules for addition and multiplication of quaternions. Assume we have two quaternions, p and q, defined as follows:

#### Quaternions p and q

$$p = a + bi + cj + dk$$

$$q = r + si + tj + uk$$

Definition of quaternions p and q for later use.

Addition of p and q is fairly intuitive:

#### Addition of Quaternions p and q

$$p + q = (a + r) + (b + s)i + (c + t)j + (d + u)k$$

Quaternion addition closely resembles vector addition. Corresponding coefficients are added to yield the sum quaternion. This operation is associative and commutative.

The dot product and magnitude of a quaternion also closely resemble those operations for vectors. Note that a **unit quaternion** is a quaternion with magnitude 1 under this definition:

Dot (Inner) Product of p and q

$$p \cdot q = ar + bs + ct + du$$

The dot product of quaternions is analogous to the dot product of vectors.

Magnitude of Quaternion p

$$||p||^2 = p \cdot p = a^2 + b^2 + c^2 + d^2$$

As with vectors, the square of the magnitude of p is the dot product of p with itself.

Multiplication, however, is not, due to the definitions of i, j, and k:

Multiplication of Quaternions p and q

$$pq = (ar - bs - ct - du) + (br + as + cu - dt)i + (cr + at + ds - bu)j + (dr + au + bt - cs)k$$

This result can be confirmed by carrying out long multiplication of p and q. There is no analog in vector arithmetic for quaternion multiplication.

Quaternion multiplication also has two equivalent matrix forms which will become relevant later in the derivation of the alignment method:

Multiplication of Quaternions p and q, Matrix Forms

$$pq = \begin{bmatrix} a & -b & -c & -d \\ b & a & -d & c \\ c & d & a & -b \\ d & -c & b & a \end{bmatrix} q$$

$$qp = \begin{bmatrix} a & -b & -c & -d \\ b & a & d & -c \\ c & -d & a & b \\ d & c & -b & a \end{bmatrix} q$$

Note that quaternions can be represented as column vectors with the imaginary components omitted. This allows vector notation to be used for many quaternion operations, including multiplication. The quaternion  $a + bi + cj + dk$ , for example, may be represented by a column vector of the form  $[a, b, c, d]$ .

These useful properties of quaternion multiplication can be derived easily using the matrix form for multiplication, or they can be proved by carrying out the products:

**Some properties of Quaternion Multiplication**

$$\begin{aligned} qp \cdot qr &= (q \cdot q)(p \cdot r) \\ (pq)(pq) &= (p \cdot p)(q \cdot q) \text{ (magnitude of product is product of magnitudes)} \\ (pq) \cdot r &= p \cdot (rq^*) \end{aligned}$$

Some useful properties.  $q^*$  is the quaternion conjugate,  $a-bi-cj-dk$

## Quaternions and Three-Dimensional Rotations

A number of different methods exist for denoting rotations of rigid objects in three-dimensional space. These are introduced in [a module on protein kinematics](#). **Unit quaternions** represent a rotation of an angle around an arbitrary axis. A rotation by the angle  $\theta$  about an axis represented by the unit vector  $\mathbf{v} = [x, y, z]$  is represented by a unit quaternion:

Unit Quaternion and Rotation

$$q = \cos \frac{\theta}{2} + \sin \frac{\theta}{2} (xi + yj + zk)$$

This unit quaternion represents a rotation of  $\theta$  about the axis defined by unit vector  $\mathbf{v} = [x, y, z]$ .

Like rotation matrices, quaternions may be composed with each other via multiplication. The major advantage of the quaternion representation is that it is more robust to **numerical instability** than orthonormal matrices. Numerical instability results from the fact that, because computers use a finite number of bits to represent real numbers, most real numbers are actually represented by the nearest number the computer is capable of representing. Over a series of floating point operations, the error caused by this inexact representation accumulates, quite rapidly in the case of repeated multiplications and divisions. In manipulating orthonormal transformation matrices, this can result in matrices that are no longer orthonormal, and therefore not valid rigid transformations. Finding the "nearest" orthonormal matrix to an arbitrary matrix is not a well-defined problem. Unit-length quaternions can accumulate the same kind of a numerical error as rotation matrices, but in the case of quaternions, finding the nearest unit-length quaternion to an arbitrary quaternion is well defined. Additionally, because quaternions correspond more directly to the axis-angle representation of

three-dimensional rotations, it could be argued that they have a more intuitive interpretation than rotation matrices. Quaternions, with four parameters, are also more memory efficient than 3x3 matrices. For all of these reasons, quaternions are currently the preferred representation for three-dimensional rotations in most modeling applications.

Vectors can be represented as purely imaginary quaternions, that is, quaternions whose scalar component is 0. The quaternion corresponding to the vector  $v = [x, y, z]$  is  $q = xi + yj + zk$ .

We can perform rotation of a vector in quaternion notation as follows:  
Rotation Using Unit Quaternions

$$\begin{aligned}r &= xi + yj + zk \\q &= a + bi + cj + dk \\q^* &= a - bi - cj - dk\end{aligned}$$

$$r' = qrq^*$$

In this figure,  $r$  is the vector  $[x, y, z]$  in quaternion form,  $q$  is a unit (rotation) quaternion,  $q^*$  is the conjugate of  $q$ , and  $r'$  is  $r$  after the rotation has been performed.

## Optimal Alignment with Quaternions

The method presented here is from Berthold K. P. Holm, "Closed-form solution of absolute orientation using unit quaternions." Journal of the

Optical Society of America A, 4:629-642.

The alignment problem may be stated as follows:

- We have two sets of points (atoms) A and B for which we wish to find an optimal alignment, defined as the alignment for which the root mean square difference between each point in A and its corresponding point in B is minimized.
- We know which point in A corresponds to which point in B. This is necessary for any RMSD-based method.

As for the case of rotation matrices, the translational part of the alignment consists of making the centroids of the two data sets coincide. To find the optimal rotation using quaternions, recall that the dot product of two vectors is maximized when the vectors are in the same direction. The same is true when the vectors are represented as quaternions. Using this property, we can define a quantity that we want to maximize (proof [here](#)):

The Objective Function for Rotational Alignment (Quaternion Form)

$$\sum_{i=1}^n (qa'_i q^* \cdot b'_i)$$

We want to  
find the  
rotation on  
set A that  
maximizes  
the sum of  
the dot  
products of  
the rotated  
vectors of A  
with the  
vectors of B,  
all expressed  
as offsets

from the set  
centroids.

Equivalently, using the last property from the section "Introduction to quaternions", we get:

$$\sum_{i=1}^n (qa'_i) \cdot (b'_i q)$$

The objective  
restated.

Now, recall that quaternion multiplication can be represented by matrices, and that the quaternions a and b have a 0 real component:

$$qa'_i = \begin{bmatrix} 0 & -a'_{i,x} & -a'_{i,y} & -a'_{i,z} \\ a'_{i,x} & 0 & a'_{i,z} & -a'_{i,y} \\ a'_{i,y} & -a'_{i,z} & 0 & a'_{i,x} \\ a'_{i,z} & a'_{i,y} & -a'_{i,x} & 0 \end{bmatrix} q = A' q$$

$$b'_i q = \begin{bmatrix} 0 & -b'_{i,x} & -b'_{i,y} & -b'_{i,z} \\ b'_{i,x} & 0 & -b'_{i,z} & b'_{i,y} \\ b'_{i,y} & b'_{i,z} & 0 & -b'_{i,x} \\ b'_{i,z} & -b'_{i,y} & b'_{i,x} & 0 \end{bmatrix} q = B' q$$

These substitutions will be used to restate the  
function to be maximized.

Using these matrices, we can derive a new form for the objective function:

$$\begin{aligned} & \sum_{i=1}^n (A'_i q) \cdot (B'_i q) \\ & \sum_{i=1}^n q^T A_i^{T'} B'_i q \\ & q^T \left( \sum_{i=1}^n A_i^{T'} B'_i \right) q \\ & q^T \left( \sum_{i=1}^n N_i \right) q = q^T N q \end{aligned}$$

The third step follows because each term in the sum is multiplied on the left and right by q, so the q factors can be moved outside the sum. The

fourth step simply renames the sum of matrix products to a single matrix, N, based on which we can find q.

where:

$$N_i = A_i^{T'} B'_i$$

$$N = \sum_{i=1}^n N_i$$

Now the  
problem is  
stated in  
terms of a  
matrix  
product  
optimization

.

The quaternion that maximizes this product is the eigenvector of N that corresponds to its most positive eigenvalue (proof [here](#)). The eigenvalues can be found by solving the following equation, which is quartic in lambda:

$$\det(N - \lambda I) = 0$$

I is the 4x4  
identity matrix.

This quartic equation can be solved by a number of standard approaches. Finally, given the maximum eigenvalue lambda-max, the quaternion corresponding to the optimal rotation is the eigenvector v:

$$(N - \lambda_{max}I)v = 0$$

This equation can be solved to find the optimal rotation.

A closed-form solution to this equation for  $v$  can be found by applying techniques from linear algebra. One possible algorithm, based on constructing a matrix of cofactors, is presented in appendix A5 of the source paper [3].

In summary, the alignment algorithm works as follows:

- Recalculate atom coordinates as displacements from the centroid of each molecule. The optimal translation superimposes the centroids.
- Construct the matrix  $N$  based on matrices  $A$  and  $B$  for each atom.
- Find the maximum eigenvalue by solving the quartic eigenvalue equation.
- Find the eigenvector corresponding to this eigenvalue. This vector is the quaternion corresponding to the optimal rotation.

This method appears computationally intensive, but has the major advantage over other approaches of being a closed-form, unique solution.

## Intramolecular Distance and Related Measures

RMSD and lRMSD are not ideally suited for all applications. For example, consider the case of a given conformation  $A$ , and a set  $S$  of other conformations generated by some means. The goal is to estimate which conformations in  $S$  are closest in potential energy to  $A$ , making the assumption that they will be the conformations most structurally similar to  $A$ . The lRMSD measure will find the conformations in which the overall average atomic displacement is least. The problem is that if the quantity of

interest is the potential energy of conformations, not all atoms can be treated equally. Those on the outside of the protein can often move a fair amount without dramatically affecting the energy. In contrast, the core of the molecule tends to be more compact, and therefore a slight change in the relative positions of a pair of atoms could lead to overlap of the atoms, and therefore a completely infeasible structure and high potential energy. A class of distance measures and pseudo-measures based on **intramolecular** distances have been developed to address this shortcoming of RMSD-based measures.

Assume we wish to compare two conformations P and Q of a molecule with N atoms. Let  $p_{ij}$  be the distance between atom i and atom j in conformation P, and let  $q_{ij}$  be the same distance for conformation Q. Then the intramolecular distance is defined as

$$\sqrt{\frac{1}{N \cdot (N + 1)} \sum_{i < j} (p_{ij} - q_{ij})^2}$$

Intra-molecular  
distance  
(dRMSD)

One of the main computational advantages of this class of approaches is that we do not have to compute the alignment between P and Q. On the other hand, for this metric we need to sum over a quadratic number of terms, whereas for RMSD the number of terms is linear in the number of atoms. Approximations can be made to speed up this computation, as shown in [7]. Also, the intramolecular distance measure given above, which is sometimes referred to as the dRMSD, is subject to the problem that pairs of atoms most distant from each other are the ones that contribute the greatest amount to their measured difference.

An interesting open problem is to come up with physically meaningful molecular distance metric that allows for fast nearest neighbor

computations. This can be useful for, for example, clustering conformations. One proposed method is the **contact distance**. Contact distance requires constructing a **contact map** matrix for each conformation indicating which pairs of atoms are less than some threshold separation. The distance measure is then a measure of the difference of the contact maps.

### Contact Distance

$$C_{ij} = \begin{cases} 1 & \text{if } r_{ij} < r_c \\ 0 & \text{otherwise} \end{cases}$$

$r_c$  = contact cutoff distance

$$q(A, B) = \frac{\sum_{i < j} C_{ij}^a C_{ij}^b}{\max(\sum_{i < j} C_{ij}^a, \sum_{i < j} C_{ij}^b)}$$

$$D(A, B) = 1 - q(A, B)$$

Contact maps (C) are calculated for each structure, and the differences in these contact maps used to define a distance D.

Other distance measures attempt to weight each pair in the dRMSD based on how close the atoms are, with closer pairs given more weight, in keeping with the intuition that small changes in the relative positions of nearby atoms are more likely to result in collisions. One such measure is the normalized **Holm and Sander Score**.

### Holm and Sander Distance

$$D(A, B) = \sum_{i < j} \frac{|r_{ij}^a - r_{ij}^b|}{r_{ij}^a + r_{ij}^b} e^{-(r_{ij}^a + r_{ij}^b)^2 / 4r_0^2}$$

This distance function is weighted to accentuate the

importance of differences in structures that are relatively close to each other. These are the contacts most likely to affect the potential energy of the structure.

This score is technically a **pseudo-measure** rather than a measure because it does not necessarily obey the **triangle inequality**.

The definition of distance measures remains an open problem. For reference on ongoing work, see articles that compare several methods, such as [5].

### **Recommended Reading:**

The first two papers are the original descriptions of the Kabsch Algorithm, and use rotations represented as orthonormal matrices to find the correct rotational transformation. Many software packages use this alignment method. The third and fourth papers use quaternions. The alignment method presented in the previous section comes from the third paper:

- W. Kabsch. (1976). [A Solution for the Best Rotation to Relate Two Sets of Vectors](#). Acta Crystallographica, 32, 922-923.
- W. Kabsch. (1978). [A Discussion of the Solution for the Best Rotation to Relate Two Sets of Vectors](#). Acta Crystallographica, 34, 827-828.
- Berthold K. P. Horn. (1986). [Closed-form solution of absolute orientation using unit quaternions](#). Journal of the Optical Society of America, 4:629-642.
- E. A. Coutsiias and C. Seok and K. A. Dill. (2004). [Using quaternions to calculate RMSD](#). Journal of Computational Chemistry, 25, 1849-1857.
- Wallin, S., J. Farwer and U. Bastolla. (2003). [Testing similarity measures with continuous and discrete protein models](#). Proteins, 50:144-157.

## Protein Classification, Local Alignment, and Motifs

### Topics in this Module

- [Applications of Molecular Distance Measures](#)
  - [Protein Classification](#)
  - [Protein Alignment](#)
- [Local Matching: Geometric Hashing, Pose Clustering, and Match Augmentation](#)

In a [previous module](#), the topic of comparing and quantifying the distance between different conformations of a given molecule was explored. Structure-based comparison is also of interest for distinct proteins, which lack the atom-by-atom correspondence necessary for RMSD calculations. In this case, an alignment is performed either based on amino acid sequence or on three-dimensional structure, and the subset of atoms successfully aligned are used as the basis for calculating conformational distance. Computing distances among entire proteins by doing a global alignment of their structures is useful for protein classification.

### Protein Classification

Protein classification is motivated by the notion of "descriptive biology". When faced with tremendous amounts of highly complex data, such as with the set of all proteins, one way to understand the data is by classification: the act of associating or grouping proteins into classes using certain criteria. One such criterion is protein sequence identity, where sequential similarity led to the development of phylogenetic trees and multiple sequence analyses. The same is done in protein structure classification, where the effort is to identify groups of similar proteins, with the hope that this will yield information about their biochemical function and biological purpose.

Proteins are classified by simultaneously applying a number of criteria, including sequence homology (evolutionary relatedness), function, folding motifs, structural features, and so on. The resulting hierarchies and clusters of protein structures provide a notion of the distance between two proteins

and their structures. A couple of popular classification schemes are linked below. Note that a fair amount of manual annotation and classification was necessary to build these systems.

## Protein Alignment

The core computational problem of protein classification, using sequence or structure, is the problem of comparing two proteins. For structural classification, one method for comparison is **structural alignment**, which identifies an ideal superimposition between two protein structures, in order to compare them.

SSAP, Dali, Foldminer, Lock, and Geometric Hashing [\[link\]](#) are algorithms which have been designed in part to align whole protein structures. Despite differences in algorithmic approach, all of these algorithms essentially evolved from the need to assign the best possible correlation between points in one structure and points in another. The problem of finding the optimal alignment is polynomial in the number of atoms in biological data, where we are assured that atoms cannot fall within a certain distance to each other (Van der Waals forces enforce this), but without this constraint the problem is exponential.

Protein alignment has been used for the classification and comparison of proteins in many existing algorithms. These include:

- [Dali](#) is a structural comparison algorithm based on pairwise distance matrices. Dali uses patterns of residue contacts, similar to contact maps described above in the intramolecular distances section, in order to align structures. The alignments are found using a randomized (Monte Carlo) search.
- [FoldMiner and LOCK 2](#). FoldMiner finds protein structures similar to an input structure by performing alignment the query structures secondary structure elements with proteins in its database using the LOCK 2 algorithm. LOCK 2 uses a combination of geometric hashing [\[link\]](#) and dynamic programming to optimize the alignments of secondary structure elements of different proteins. Once a set of

alignments to similar structures are found, motifs consisting of similar secondary structure arrangements are constructed and used to refine the similarity search.

- [Sequential Structure Alignment Program \(SSAP\)](#). Given two protein structures, SSAP returns a structural alignment.

## **Protein Classification**

Once alignment algorithms have been implemented, it is possible to explore different classifications of proteins. Naturally, it would be intuitive to classify proteins solely according to their structure, but much richer data is available as well. Current classifications of proteins integrate sequence and structure information in order to maximize their utility. These include the following:

- [SCOP \(Structural Classification of Protein\)](#) is a database of all proteins whose structures have been determined, organized by family (evolutionary relationship), superfamily (structural and functional similarity), and fold (similar secondary structure, with similar arrangement and topological connections). SCOP was constructed largely by manual inspection and annotation.
- [CATH Protein Structure Classification](#) is another database of protein structures, organized by Class (secondary structure content), Architecture (orientation of secondary structures), Topology (overall shape and connectivity), and Homologous Superfamily (evolutionary relationship inferred based on sequence and structure similarity). CATH uses SSAP (the Sequential Structural Alignment Program, a secondary structure element-based method) for structural comparison.

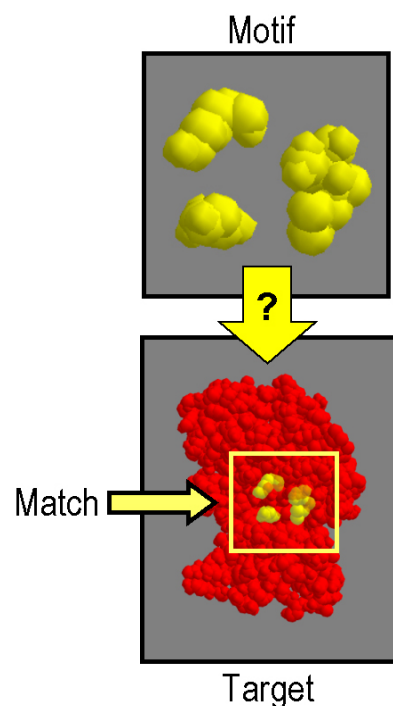
The biological purpose of designing Global Structural Alignment algorithms was the identification, classification, and ultimately prediction, of protein function, under the hypothesis that protein structure dictates protein function. Simultaneously, it was realized that small changes to proteins could lead to massive changes in function, or nothing at all. This suggested that global alignment and global structure comparison (as well as

global sequence alignment and global sequence comparison) should not be the only tools used for function prediction.

In particular, it was realized that active sites, clusterings of amino acids on the surface of proteins and a tiny minority of the entire protein, were often strongly related to protein function. In a continuation of the effort to predict protein function through structural comparison, algorithms were developed to compare functionally relevant substructures of proteins. We refer to these algorithms collectively as local structure alignment algorithms.

### Local Matching: Geometric Hashing, Pose Clustering and Match Augmentation

Algorithms for local structure alignment address the similar computational problem of selecting a correspondence between a **motif**, a tiny substructure of a protein, often between 3 and 20 amino acids, and a **target**, a full protein structure. Once a correspondence has been established, the "distance" of the motif to the identified part of the target is measured using IRMSD.



Some algorithms for local structure alignment are based on pattern matching algorithms. Pattern matching algorithms seek target substructures called matches which have maximal geometric similarity to the motif. An excellent example of this type of algorithm is Geometric Hashing [\[link\]](#), a very flexible framework for geometric pattern matching under noisy constraints. Geometric Hashing [\[link\]](#), has been adapted to alignment by atom position [\[link\]](#), by backbone C-alpha [\[link\]](#), multiple structural alignment [\[link\]](#), and alignment of hinge-bending and flexible protein models [\[link\]](#). Other algorithms for substructural alignment include JESS [\[link\]](#), PINTS [\[link\]](#), webFEATURE [\[link\]](#), and pvSOAR [\[link\]](#). The description below concentrates on the work developed in [\[link\]](#).

## Motifs

A motif  $S$  is a set of  $m$  points  $s_1, \dots, s_m$  in three dimensions, whose coordinates are taken from backbone and side-chain atoms. Each **motif point**  $s_i$  in the motif has an associated **rank**  $p(s_i)$ , a measure of the functional significance of the motif point. Each  $s_i$  also has a set of alternate amino acid **labels**  $l(s_i)$  in  $\{\text{GLY}, \text{ALA}, \dots\}$ , which represent residues this amino acid has mutated to during evolution. Labels permit our motifs to simultaneously represent many homologous active sites with slight mutations, not just a single active site. In this paper, we obtain labels and ranks using Evolutionary Trace (ET) [\[link\]](#),[\[link\]](#).

Other motifs have been designed with other approaches. Motifs have been composed of points on the Connolly surface [\[link\]](#) representing electrostatic potentials [\[link\]](#), of hinge-bending sets of points in space [\[link\]](#), of sets of "pseudo-centers" representing protein-ligand interactions [\[link\]](#), or of points taken from atom coordinates with evolutionary data [\[link\]](#), [\[link\]](#), to name a few. Depending on how motif points are defined, they have different labels associated with them and these labels need to be taken into account when comparing motifs.

## Protein Function Prediction

Local structure comparison algorithms mainly target the biological problem of Protein Function Prediction. Ideally, a function prediction pipeline should address the following subproblems:

- The design of effective motifs
- The efficient identification of matches
- The effective filtering of matches

Suppose for a moment that we have hand-designed motifs. In the next section, we concentrate on an efficient algorithm for local structure comparison.

## Identification of Matches

Many such algorithms exist, but differ fundamentally in that they are optimized for comparing different types of motifs. There are algorithms for comparing graph-based motifs [\[link\]](#), algorithms for finding catalytic sites [\[link\]](#), and the seminal Geometric Hashing framework [\[link\]](#) which can search for many types of motifs, including motifs based on atom position [\[link\]](#), points on Connolly face centers [\[link\]](#), catalytic triads [\[link\]](#), and flexible protein models [\[link\]](#). **Match Augmentation** (MA) [\[link\]](#) is described below.

MA compares a motif  $S$  to a target  $T$ , a protein structure encoded as  $|T|$  *target points*:  $T = \{t_1, \dots, t_{|T|}\}$ , where each  $t_i$  is taken from atom coordinates, and labeled  $l(t_i)$  for the amino acid to which  $t_i$  belongs. A match  $M$  is a bijection correlating all motif points in  $S$  to a subset of  $T$  of the form  $M = \{(s_{a_1}, t_{b_1}), (s_{a_2}, t_{b_2}) \dots (s_{a_{|S|}}, t_{b_{|S|}})\}$ . Referring to the Euclidean distance between points  $a$  and  $b$  as  $\|a - b\|$ , an acceptable match requires:

**Criterion 1**  $\forall i, s_{a_i}$  and  $t_{b_i}$  are biologically compatible:  $l(t_{b_i}) \in l(s_{a_i})$ .

**Criterion 2** LRMSD alignment, via rigid transformation  $A$  of  $S$ , causes  $\forall i, \|A(s_{a_i}) - t_{b_i}\| < \epsilon$ , our threshold for geometric similarity.

MA takes as input a motif  $S$  and a target  $T$ . MA outputs the match with smallest LRMSD among all matches that fulfill the criteria. Partial matches correlating subsets of  $S$  to  $T$  are rejected. By establishing a threshold for acceptable geometric similarity, the second criterion causes MA to return match LRMSDs bounded above by  $\epsilon$ . We find that  $\epsilon = 7 \text{ \AA}$  permits the identification of structurally distant matches when no matches with lower LRMSD exist, while still efficiently identifying matches with high structural similarity.

## Match Augmentation

Rank information prioritizes motif data and MA was designed in a prioritized fashion, where correspondences with higher ranked points are identified first. MA is composed of two parts: **Seed Matching** and **Augmentation**. The purpose of Seed Matching is to identify a match for the **seed**  $S' = \{s_1, s_2, s_3\}$ , the three highest ranked motif points. The  $k$  lowest LRMSD **seed matches** are passed to Augmentation to be iteratively expanded into matches for the remaining motif points, in descending rank order. Augmentation outputs the match with smallest LRMSD.

### Seed Matching

Seed Matching identifies all sets of 3 target points  $T' = \{t_A, t_B, t_C\}$  which fulfill our matching criteria with the highest ranked 3 motif points,  $S' = \{s_1, s_2, s_3\}$ . In this stage, we represent the target as a geometric graph with colored edges. There are exactly three unordered pairs of points in  $S'$ , and

we name them red, blue and green. In the target, if any pair of target points  $t_i, t_j$  fulfills our first two criteria with either red, blue or green, we draw a corresponding red blue or green edge between  $t_i, t_j$  in the target. Once we have processed all pairs of target points, we find all three-colored triangles in T. These are the Seed Matches, a set of three-point correlations to S' that we sort by lRMSD and pass to Augmentation.

### **Augmentation**

Augmentation is an application of depth first search that begins with the list of seed matches. Assuming that there are more than four motif points, we must find correspondences for the unmatched motif points within the target. Interpret the list of seed matches as a stack of partially complete matches. Pop off the first match, and considering the lRMSD alignment of this match, plot the position p of the next unmatched motif point  $s_i$  relative to the aligned orientation of the motif. In the spherical region V around p, identify all target points  $t_i$ , compatible with  $s_i$ , inside V. Now compute the lRMSD alignment of all correlated points, include the new correlation  $(s_i, t_i)$ . If the new alignment satisfies our first two criteria and there are no more unmatched motif points, put this match into a heap which maintains the match with smallest lRMSD. If there are more unmatched motif points, put this partial match back onto the stack. Continue to test correlations in this manner, until V contains no more target points that satisfy our criteria. Then, return to the stack, and begin again by popping off the first match on the stack, repeating this process until the stack is empty.

### **Filtering Matches**

Structural similarity is important to functional annotation only if a strong correlation exists between identifiably significant structural similarity and functional similarity. However, the existence of a match alone does not guarantee functional similarity. lRMSD can be a differentiating factor. If matches of homologous proteins represent statistically significant structural similarity over what is expected by random chance, we could differentiate

on IRMSD, as long as we can evaluate the statistical significance of the IRMSD of a match.

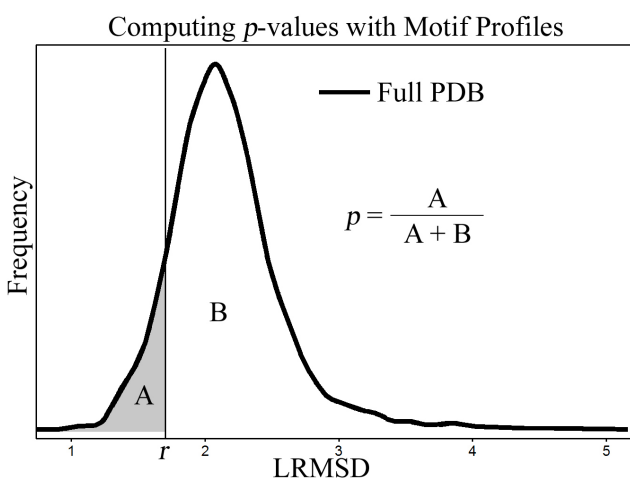
BLAST [\[link\]](#) first calculated the statistical significance of sequence matches with a combinatorial model of the space of similar sequences. Determining the statistical significance of structural matches has also been attempted. Modeling was applied for the PINTS database [\[link\]](#) to estimate the probability of a structural match given a particular LRMSD. An artificial distribution was parameterized by motif size and amino acid composition in order to fit a given data set, and the p-value is calculated relative to that distribution. Another approach was taken in the algorithm JESS [\[link\]](#), using comparative analysis to generate a significance score relative to a specific population of known motifs. Both methods have some disadvantages. The artificial models of PINTS are not parameterized by the geometry of motifs, and, all else equal, produce identical distributions for motifs of different geometry. JESS, on the other hand, is dependent on a set of known motifs; should this set change, all significance scores would have to be revised.

Local structural alignment methods operate on the assumption that local structural and chemical similarity implies functional similarity. A statistical model has been developed that can be used to identify the degree of similarity sufficient to follow this implication. Given a match  $m$  with IRMSD  $r$  between motif  $S$  and target  $T$ , exactly one of two hypotheses must hold:

- $H_0$ :  $S$  and  $T$  are structurally dissimilar
- $H_A$ :  $S$  and  $T$  are structurally similar

The proposed statistical model implements this measurement by computing a **motif profile**. Motif profiles are frequency distributions (see Figure below) of match IRMSDs between  $S$  and the entire Protein Data Bank (PDB) [\[site\]](#), which is essentially a large set of functionally unrelated proteins. A motif profile is essentially a histogram, where the vertical axis measures the number of matches at each specific IRMSD, which is measured on the horizontal axis. Motif profiles provide very complete information about matches typical of  $H_0$ . If we suspect that a match  $m$  has IRMSD  $r$  indicative of functional similarity, we can use the motif profile to

determine the probability  $p$  of observing another match  $m'$  with smaller LRMSD by computing the volume under the curve to the left of  $r$ , relative to the entire volume. The probability  $p$ , often referred to as the  $p$ -value, is the measure of statistical significance. With a standard of statistical significance  $\alpha$ , if  $p$  is less than  $\alpha$ , then we say that the probability of observing a match  $m'$  with LRMSD  $r'$  less than  $r$  is so low that we reject the null hypothesis in favor of the alternative hypothesis. This process means that if a match  $m$  with LRMSD  $r$  has a  $p$ -value exactly equal to  $\alpha$ , then this LRMSD is the highest LRMSD for which our statistical model predicts that  $m$  identifies structural and chemical similarity sufficient to imply functional similarity. Matches with this property are considered to be statistically significant.



The use of motif profiles.

## Designing Effective Motifs

Given a motif representing a specific biochemical function, the task of any algorithm seeking to predict protein functions is to identify other proteins with similar biological functions. This is immediately affected by how well

the motif represents the biological function - protein structures are flexible, dynamic objects, and any computational representation of these objects is likely to be inaccurate in some manner. In particular, since it is the computational representation that is compared, the fidelity of representation is essential for the effectiveness of the structural comparison approach.

An effective motif must simultaneously fulfill two criteria:

- The motif must be sensitive
- The motif must be specific

Motifs must maintain geometric and chemical similarity, in respect to the characteristics compared by the comparison algorithm, to functional homologs. This ensures that algorithms searching for the motif identify proteins with similar function.

Sensitivity is measured as the proportion of acceptable matches to functional homologs relative to the total number of functional homologs. Specific motifs must maintain geometric and chemical dissimilarity, in respect to the characteristics compared by the comparison algorithm, to functionally unrelated proteins. This ensures that algorithms searching for the motif accidentally identify less matches to functionally unrelated proteins. Specificity is measured as the proportion of rejected matches to functionally unrelated proteins, relative to all functionally unrelated proteins. The ideally effective motif is 100% sensitive and 100% specific.

Under most conditions, effective motifs have been designed by hand. However, a few recent studies exist which examine the relationship between elements of the motif and the sensitivity and specificity of the motif. These studies resulted in the design of MultiBind [\[link\]](#), an algorithm which identifies conserved binding patterns among functionally homologous active sites, in order to generate motifs which represent only conserved binding patterns. Another study of motif design produced Geometric Sieving [\[link\]](#), an algorithm for maximizing the Geometric Uniqueness of motifs from functionally unrelated proteins.

Putting together effective motif design, an efficient matching algorithm, and a statistical scoring of the results can lead to an automated functional

annotation pipeline for proteins.

### **Recommended Reading:**

- Chen, B.Y. (2005). [Algorithms for Structural Comparison and Statistical Analysis of 3D Protein Motifs](#). Proceedings of the Pacific Symposium on Biocomputing 2005, pp. 334-345.
- Chen, B.Y. (2006). [Geometric Sieving: Automated Distributed Optimization of 3D Motifs for Protein Function Prediction](#). Proceedings of the Tenth Annual Conference on Research in Computational Molecular Biology (RECOMB 2006), pp. 500-512.
- Yao, H., Kristensen, D., Mihalek, I., Sowa, M., Shaw, Ch., Kimmel, M., Kavraki, L. E., and Lichtarge, O. (2003). [An Accurate, Sensitive, and Scalable Method to Identify Functional Sites in Protein Structures](#). Journal of Molecular Biology, 325:255-261.
- David M. Kristensen, Brian Y. Chen, Viacheslav Y. Fofanov, R. Matthew Ward, A. Martin Lisewski, Marek Kimmel, Lydia E. Kavraki, and Olivier Lichtarge. (2006). [Recurrent Use of Evolutionary Importance for Functional Annotation of Proteins Based on Local Structural Similarity](#). Protein Science, in press.

## Dimensionality Reduction Methods for Molecular Motion

### Topics in this Module

- [Introduction](#)
- [Dimensionality Reduction](#)
  - [Principal Components Analysis](#)
    - [PCA of conformational data](#)
  - [Non-Linear Methods](#)
    - [Isometric Feature Mapping \(Isomap\)](#)

### Introduction

The study of many biological processes at the molecular level involves understanding how biological molecules (especially proteins) behave dynamically. The three-dimensional shape of these molecules, which we call a **conformation**, usually determines the chemical action they perform. Both the stable (also called **native**) shape of a biomolecule and dynamical deviations from it are important to understand how it interacts with other molecules such as pharmaceutical drugs or other complexes. For these reasons, understanding the main shapes and motions of these molecules is of utmost importance.

Current structural biology experimental methods are restricted in the amount of information they can provide regarding protein motions because they were designed mainly to determine the three-dimensional static representation of a molecule. For this reason, **in silico** methods (i.e., run in a computer) are used to extensively sample protein conformations. As a summary, the most popular methods to gather protein conformations are:

- **X-Ray Crystallography.** The most established and accurate method of determining the three-dimensional structure of a protein is [X-ray crystallography](#). This technique is based on the collection of diffraction data generated by exposing a protein crystal to an X-ray beam. The main limitation of this experimental technique is that it is necessary to

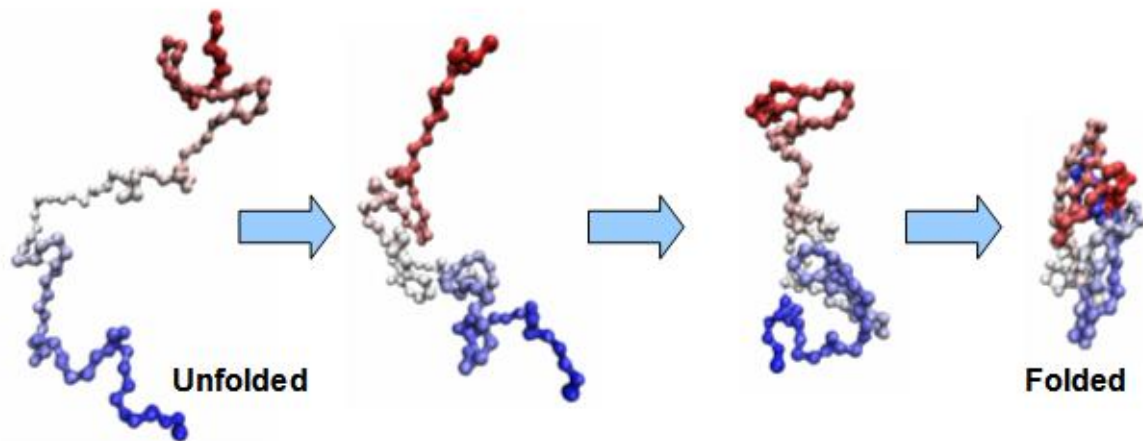
obtain protein crystals in order to collect experimental data.

Unfortunately, creating protein crystals is a very lengthy and laborious process which is not always successful.

- **Nuclear Magnetic Resonance (NMR).** The second most common method of determining the structure of a protein is [NMR](#) {Wuthrich, 1986 #184}. This method uses a spectroscopy approach to collect the experimental data necessary for structure determination. This method is in general not as accurate as X-ray crystallography and its use is limited to small and medium-sized proteins. However, it provides useful information about protein dynamics directly and avoids some of the problems of X-ray crystallography such as protein crystallization.
- **In silico sampling.** An alternative to using experimental methods to derive structural data is using computational methods such as Molecular Dynamics (MD) or Monte Carlo (MC) simulations, or other forms of computational sampling. In fact, computational methods are used to augment existing experimental data since MD simulations typically start from a three-dimensional protein structure determined by X-ray crystallography or NMR. MD uses an empirical **force field** to approximate the potential energy of a protein shape. Once a force field model has been chosen, the time evolution of the system is determined by numerically solving the resulting equations of motion. One of the main disadvantages of MD is that it is very computationally expensive, making it impossible (with current technology) to run all-atom simulations of big proteins for time-scales relevant to the majority of biological processes. Nevertheless, simulations can provide us with invaluable data since they are the only method of observing proteins in “real time”. Recently, the development of so-called **coarse-grained** models (which model a group of atoms as a single entity) has allowed the sampling of longer times. MD is a good data source for sampling purposes because it can provide a large number of conformations of a molecule. For an introduction to Molecular Dynamics simulations, please refer to [\[6\]](#).

The computations required to simulate protein motion in silico are very expensive and involve non-trivial force field evaluations, as explained above. These simulations provide us with the (x,y,z) positions of all atoms in the molecule; for a molecule with N atoms, this amounts to 3N numbers

per conformation. For interestingly sized molecules (such as proteins) the number of atoms is large and thus the dimensionality of the obtained data is extremely high. That is, a conformation sample for a protein with  $N$  atoms can be thought of as a  $3N$ -dimensional point.



Protein folding is an important biological process. Individual atom motions are very complex, but the overall process seems almost one-dimensional, that is, following a direction from "unfolded" to "folded".

However, many biological processes are known to be very structured at the molecular level, since the constituent atoms self-organize to achieve their bio-chemical goal. An example of such a process is **protein folding**, the process by which a protein achieves its thermodynamically-stable three-dimensional shape to perform its biological function (a depiction of a protein folding process is shown in figure 1). To study such processes based on data gathered through simulations, there is a need to "summarize" the high-dimensional conformational data. Simply visualizing the time-series of a moving protein as produced by simulation packages does not provide a lot of insight into the process itself. One way of summarizing these motions is to turn conformations into a low-dimensional representation, such as a vector with very few components, that somehow give the "highlights" of the process. This data analysis process -turning high-dimensional data into low-

dimensional data for better interpretation- is called **dimensionality reduction** and is discussed next.

## Dimensionality Reduction

When molecular shapes are sampled throughout some physical-chemical process that involves the motion of the molecule, there is a need to simplify the high-dimensional (albeit redundant) representation of a molecule given as a  $3N$ -dimensional point, since it is believed that the actual degrees of freedom (DoFs) of the process are much less, as explained before. The resulting, simplified representation has to be useful to classify the different conformations along one or more "directions" or "axes" that provide enough discrimination between them.

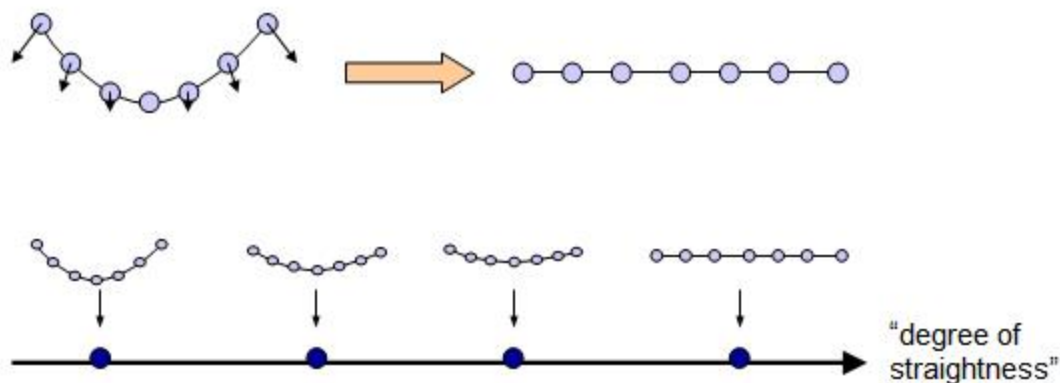
**Dimensionality Reduction** techniques aim at analyzing a set of points, given as input, and producing the corresponding low-dimensional representation for each. **The goal is to discover the true dimensionality of a data set that is only apparently high-dimensional.** There exist mathematical tools to perform automatic dimensionality reduction, based on arbitrary input data in the form of high-dimensional points (not just molecules). Although different techniques achieve their goals in different ways, and have both advantages and disadvantages, the most general definition for dimensionality reduction could be stated as:

### Dimensionality Reduction

- **INPUT:** A set of  $M$ -dimensional points.
- **OUTPUT:** A set of  $d$ -dimensional points, one for each of the input points, where  $d \ll M$ .

Some dimensionality reduction methods can also produce other useful information, such as a "direction vector" that can be used to interpolate atomic positions continuously along the main motions (like in PCA, see below). For both a general discussion and specific methodology on dimensionality reduction, refer to [7], and for more information on non-linear methods, see [8].

As a simple example of dimensionality reduction, consider the case of a bending string of beads, as depicted in figure 2. The input data has  $3 \times 7 = 21$  dimensions (if given as the  $(x,y,z)$  coordinates of each bead) but the beads always move **collectively** from the "bent" arrangement (left) to the "straight" arrangement (right). Under this simplified view, the process can be considered as one-dimensional, and a meaningful axis for it would represent the "degree of straightness" of the system. Using this axis, each string of beads can be substituted by one single number, its "coordinate" along the proposed axis. Thus, the location of a shape along this axis can quickly indicate in what stage of the bending process it is.



Sampled data from a chain of 7 beads can be given by the  $(x,y,z)$  positions of each bead, for a total of 21 "apparent" degrees of freedom. However, the process is inherently one-dimensional.

When dimensionality reduction methods are applied to molecular motion data, the goal is to find the main "directions" or "axes" collectively followed by the atoms, and the placement of each input conformation along these axes. The meaning of such axes can be intuitive or abstract, depending on the technique used and how complex the system is. We can

reword the definition of dimensionality reduction when working with molecular motion samples as:

### **Dimensionality Reduction of Molecular Motion Data**

- **INPUT:** A set of molecular conformations sampled from some physical process, given as the  $(x,y,z)$  coordinates for each atom. These are  $3N$ -dimensional points for a molecule with  $N$  atoms.
- **OUTPUT:** A set of  $d$  coordinates for each input conformation, such that  $d \ll 3N$ . These  $d$  coordinates should help classify the conformations throughout the main stages of the studied process.

Dimensionality reduction methods can be either **linear** or **non-linear**. Linear methods typically compute the low-dimensional representation of each input point by a series of mathematical operations involving linear combinations and/or linear matrix operations. Non-linear methods use either non-linear mathematics or modify linear methods with algorithmic techniques that encode the data's "curvature" (such as Isomap, explained later). Both categories of methods have advantages and disadvantages, which will become clear through the rest of this module. More information on different linear and non-linear techniques for dimensionality reduction are given in [7]. Also, a more extensive classification of dimensionality reduction techniques into several categories, and examples of different algorithms can be found navigating [this hypertext article](#).

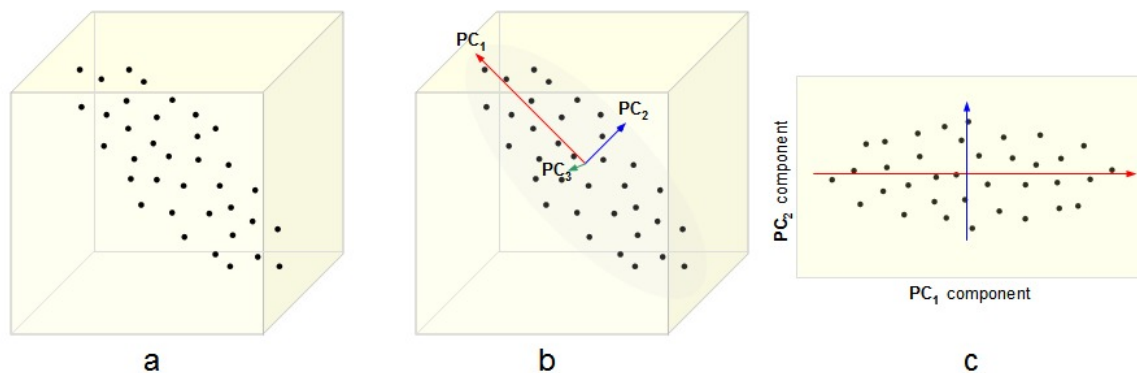
The remainder of this module describes two dimensionality reduction techniques and their application to molecular motion data. These are Principal Components Analysis (PCA), a linear method, and ISOMetric feature MAPping (Isomap), a non-linear method.

### **Principal Components Analysis**

We will start our discussion of Principal Components Analysis, or PCA, considering a very general data set of points as depicted in figure 3. Each point in this simple data set is given as a 3-dimensional vector  $(x,y,z)$  (the discussion will later be turned to the molecular motion domain, and the interpretation of such data). Even though this data set is given as 3-

dimensional points, it is obvious from the figure that the data points are distributed mostly on a two-dimensional surface. Our objective is then to find the inherent, 2-dimensional parameterization of this data set. (For a full discussion on PCA, see [Principal Components Analysis](#) by I. T. Jolliffe).

### Principal Components Analysis



An illustration of PCA. **a)** A data set given as 3-dimensional points. **b)** The three orthogonal Principal Components (PCs) for the data, ordered by variance. **c)** The projection of the data set into the first two PCs, discarding the third one.

For data points of dimensionality  $M$ , the goal of PCA is to compute  $M$  so-called **Principal Components** (PCs), which are  $M$ -dimensional vectors that are aligned with the directions of maximum variance (in the mathematical sense) of the data. These PCs have the following properties:

- The PCs are ordered by data variance. In other words, the first PC is aligned with the direction of maximum variance, the second PC in the next direction contributing to the most variance, and so on.
- The PCs form an **orthonormal basis**, that is, they are all mutually perpendicular and have unit length. This gives PCs the useful property of being **uncorrelated**.

For example, in figure 3 b), the PCs have been superimposed with the data set (PCs are drawn with different lengths to illustrate the amount of data variance they account for, but remember they are actually of unit length).

Since the PCs are orthonormal, they form a vector basis in terms of which the data set can be expressed. An alternative, equivalent view, is that the PCs become aligned with the canonical axes X,Y,Z,... if the data set is **rotated** or **rigidly transformed** so that the directions of maximum cumulative variance are made to coincide with the canonical base. For the simple example, the reader can agree that the last direction of maximum variance, the 3rd in this case, accounts for little or no data variability. It is customary, when projecting the original data set into the principal components, to **discard** the components that do not add a significant contribution to the data variance. In figure 3 c), the third component has been discarded and the projection of the data onto the first two components is shown. Discarding the least-important components is how **dimensionality reduction** is performed.

For M-dimensional input data, the Principal Components (PCs) are M-dimensional vectors and have two main uses. These are to:

- **Project the input data onto the PCs.** Taking the dot product of an input data point with any PC returns the scalar value of the projection of the point onto the PC. Since the PCs have unit length, this projection serves as the **coordinate** of the input point along the PC in question. In principle, M-dimensional input data can be projected onto its M PCs, but typically we are not interested in the lesser ones (the data set would be nicely aligned with the PCs, but we would still be using M coordinates for each point). Using just the first few PCs as a basis and computing the projections onto say, the first  $d$  PCs yields the best  $d$ -dimensional representation for each point, from a maximum variance point of view (e.g.,  $d=2$  in figure 3-c). This is the actual dimensionality reduction.
- **Interpolate or synthesize new points.** The PCs themselves point in the direction of maximum variance, as explained above. For this reason,  $PC_i$  can be used as a **direction vector** along which new points can be synthesized by choosing parameter values  $a_i$  and then producing artificial M-dimensional points by doing the linear combination  $a_1PC_1 + a_2PC_2 + \dots$ . Points synthesized in this way would lie approximately on the low-dimensional hyperplane spanned by the original data set. The projections of the original points

correspond to particular values for these new "coordinates"  $a_i$ . Being able to interpolate other points not in the original data set is a useful property that other dimensionality reduction methods do not have.

There is one detail left to introduce before going into the mechanics of computing the principal components. Notice that in figure 3-a the data set is not shown at any particular position in 3D space, that is, the data set could be centered at any point in space, not necessarily around the origin (0,0,0). However, in order to compute the principal components, compute the low-dimensional projections, and synthesize new points, the data set is assumed to be **centered** in every direction. In other words, the data set needs to have its **centroid** removed before the computations. If new points need to be synthesized using the PCs, the centroid can be added to place the newly synthesized points in the correct region of space.

To compute the principal components, let  $X$  be an  $n \times M$  matrix that contains  $n$   $M$ -dimensional data points in its columns. Furthermore, assume that the mean for all dimensions is zero, i.e., the data are centered. The goal is to find an  $M \times M$  orthonormal transformation matrix  $P$  containing the PCs, such that:

- $Y = P^T X$ , where the columns of  $Y$  are the projections onto the PCs.
- $PP^T = I$ , that is,  $P$  is orthonormal.
- $YY^T = D$ , the covariance matrix of the projected points  $Y$ , is a diagonal matrix, so that the resulting projections are uncorrelated.

The resulting covariance matrix  $YY^T$  can be re-written as:

$$YY^T = (P^T X)(P^T X)^T = P^T (XX^T)P$$

We want  $YY^T$  to be a diagonal matrix  $D$  so we can write:

$$P^T (XX^T)P = D$$

So then, by multiplying by  $P$  to the left and  $P^T$  to the right:

$$XX^T = PDP^T$$

Since  $PP^T = I$ . Note that the [Singular Value Decomposition](#) (SVD) of  $XX^T$  yields:

$$XX^T = VSW^T$$

Where  $V$  and  $W$  are the left and right eigenvectors of  $XX^T$ , and  $S$  is a diagonal matrix with the eigenvalues. But since  $XX^T$  is a symmetric matrix by construction, the left and right eigenvectors coincide, so  $W = V$ . So we can write:

$$PDP^T = VSV^T$$

This fact, together with the fact that  $P$  and  $V$  are orthonormal, means that  $P = V$  and  $D = S$  (because both  $D$  and  $S$  are diagonal). In other words, the Principal Components, or PCs, of the data set  $X$  are given by the eigenvectors of the covariance matrix  $XX^T$  of the original (centered) data. Moreover, the diagonal matrix of eigenvalues,  $S$ , is equal to the matrix  $D$ , which is the covariance of the projected points  $Y$ . Since it is a diagonal matrix, the diagonal contains the **variance** of the projected data set along each dimension. By retrieving the eigenvectors (or PCs) in order of decreasing eigenvalue, the PCs that explain the most data variance are automatically chosen before the rest. Since most SVD computations return the eigenvectors and eigenvalues in this order anyway, the resulting PCs are already sorted by variance.

It is important to note at this point that the eigenvalues actually correspond to the variance along each PC. By computing the ratio of each eigenvalue  $s_i$  to the total sum, one can obtain the fraction of total variance explained by each PC when the data is projected onto them. Subtracting the sum of variance fraction for the first  $d$  PCs from 1, we can obtain what is known as the **residual variance**  $r_d$ , that is, the amount of variance in the original data

left unexplained by discarding the PCs corresponding to the lower  $M-d$  eigenvalues:

$$r_d = 1 - \sum_{i=1}^d \frac{s_i}{\sum_{j=1}^M s_j}$$

which is a typical measure of the error made by approximating the data set using only the first  $d$  principal components.

The above derivation can be written as an algorithm fairly easily.

### **PCA algorithm**

1. Let the input data consist of  $n$  observations  $x_i$ , each of dimensionality  $M$ . Construct an  $n \times M$  matrix  $X$  of centered observations by subtracting the data mean from each point, so that  $X_{ij} = x_{ij} - \langle x \rangle_j$
2. Construct the **covariance matrix**  $C = XX^T$
3. Compute the top  $d$  eigenvalues and corresponding eigenvectors of  $C$ , for example by performing an SVD of  $C$ .
4. The first  $d$  Principal Components (PCs) of the data are given by the eigenvectors, which can be placed in a  $d \times M$  matrix  $P$ . The residual variance can be computed from the eigenvalues as explained above.
5. To project the original (centered) points into the optimal  $d$ -dimensional hyperplane, compute the dot product of each point with the PCs to obtain the projections  $y_i$ . This can be written as  $Y = P^T X$ .

PCA is very well established as a dimensionality reduction technique and efficient algorithms with guaranteed convergence for its computation are readily available. Software packages that perform SVD and PCA are freely available and trivial to use. For example Matlab has built-in commands for both, and moderately experienced C and Fortran programmers can use the popular and extremely-efficient LAPACK linear algebra package. The concepts explained in this section have not assumed any particular number of dimensions. Even though a 3D example was given at the beginning, the concepts can be lifted to deal with spaces of arbitrary dimensionality, where the input data set can have a large dimension  $M$ . PCA has the advantage over other available methods that the principal components have a direct

physical interpretation, especially when working with molecular motion data.

### PCA of conformational data

In structural bioinformatics we want to apply PCA to a set of molecular conformations, which will serve as our high-dimensional points. The input dimensionality of each point is  $3N$ , where  $N$  is the number of atoms in the molecule. We will have  $n$  such conformations, that have been gathered through some form of sampling (for example through molecular dynamics simulations), and we want to reduce the dimensionality of each "point" (conformation) for analysis purposes. The data used as input for PCA is in the form of several atomic position vectors corresponding to different structural conformations which together constitute a vector set. Each vector in the conformational vector set has dimension  $3N$  and is of the form  $[x_1, y_1, z_1, x_2, y_2, z_2, \dots, x_N, y_N, z_N]$ , where  $[x_i, y_i, z_i]$  corresponds to the Cartesian coordinates of the  $i^{\text{th}}$  atom.

However, before running the PCA procedure outlined above, all conformations need to be **aligned** with a reference structure first, as discussed in the module [Molecular Distance Measures](#). The reason why alignment is important is that most simulation packages model the exchange of heat between the molecule and a thermal bath, in the form of random velocity perturbations on the atoms. These perturbations will in general add non-zero linear and angular momenta to the molecule structure, and not all simulation packages remove them. As a result, molecular conformations that are almost identical in shape but translated/rotated with respect to each other will have significantly different coordinates, and will be considered as different  $3N$ -dimensional points. The reference structure (to which all structures should be aligned to) can be chosen from the set (e.g., the native structure) or provided from outside the set. Results may vary a little, but aligning all conformations to the same reference structure yields comparable results in general.

After all conformations have been aligned, the PCA procedure can be used exactly as detailed above. The first step is to determine the average vector

from the conformational vector set, so it can be subtracted from all conformations to build the matrix  $X$ . This is done by computing an **average conformation** that contains the average for all  $3N$  dimensions of the data set. It is important to note that this "average conformation" no longer represents a physically feasible molecule, since the Cartesian coordinates of all atoms are being averaged throughout the entire data set. When this "average conformation" is subtracted from the aligned input conformations, the "centered" data now becomes atomic **displacements, not positions**, and so building a covariance matrix for this data makes sense.

The PCs for molecular conformation data can be used for the two purposes explained before, i.e., to obtain a low-dimensional representation of each point and to synthesize (or interpolate) new conformations by following the PCs. Now, the PCs have a physical meaning: they represent the "main directions" followed by the molecule's  $3N$  degrees of freedom, or, in other words, the directions followed collectively by the atoms. Interpolating along each PC makes each atom follow a linear trajectory, that corresponds to the direction of motion that explains the most data variance. For this reason, the PCs are often called **Main Modes of Motion** or **Collective Modes of Motion** when computed from molecular motion data. Interpolating along the first few PCs has the effect of removing atomic "vibrations" that are normally irrelevant for the molecule's bigger scale motions, since the vibration directions have little data variance and would correspond to the last (least important) modes of motion. It is now possible to define a lower-dimensional subspace of protein motion spanned by the first few principal components and to use these to project the initial high-dimensional data onto this subspace. Since the PCs are **displacements** (and not positions), in order to interpolate conformations along the main modes of motion one has to start from one of the known structures and add a multiple of the PCs as perturbations. In mathematical terms, in order to produce conformations interpolated along the first PC one can compute:

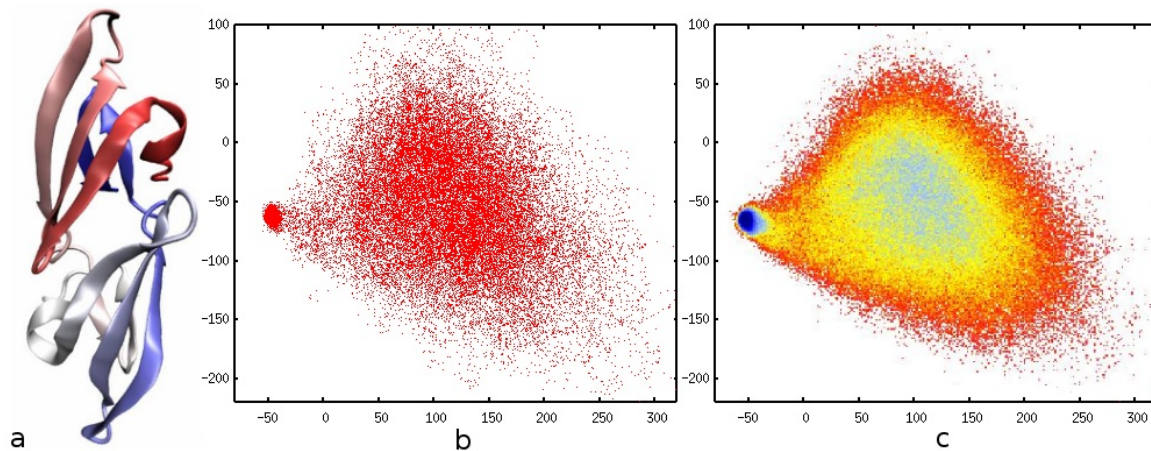
$$c + \lambda_1 PC_1$$

Where  $c$  is a molecular conformation from the (aligned) data set and the interpolating parameter can be used to add a deviation from the structure  $c$

along the main direction of motion. The parameter can be either positive or negative. However, it is important to keep in mind that large values of the interpolating parameter will start stretching the molecule beyond physically acceptable shapes, since the PCs make all atoms follow straight lines and will fairly quickly destroy the molecule's topology. A typical way of improving the physical feasibility of interpolated conformations is to subject them to a few iterations of energy minimization following some empirical force field.

Although it is possible to determine as many principal components as the number of original variables ( $3N$ ), PCA is typically used to determine the smallest number of uncorrelated principal components that explain a large percentage of the total variation in the data, as quantified by the residual variance. The exact number of principal components chosen is application-dependent and constitutes a truncated basis of representation.

As an example of the application of PCA to produce low-dimensional points for high-dimensional input, consider the Cyanovirin-N (CV-N) protein depicted in figure 4 a), corresponding to PDB code 2EZM. This protein acts as a virucidal for many viruses, such as HIV. Protein folding simulations of this protein can be used as input for a PCA analysis. A model of this protein that considers atoms at the alpha-carbon positions only has 101 such atoms, for a total of 303 degrees of freedom. Folding/unfolding simulations starting from the native PDB structure produce abundant conformation samples of CV-N along the folding reaction.

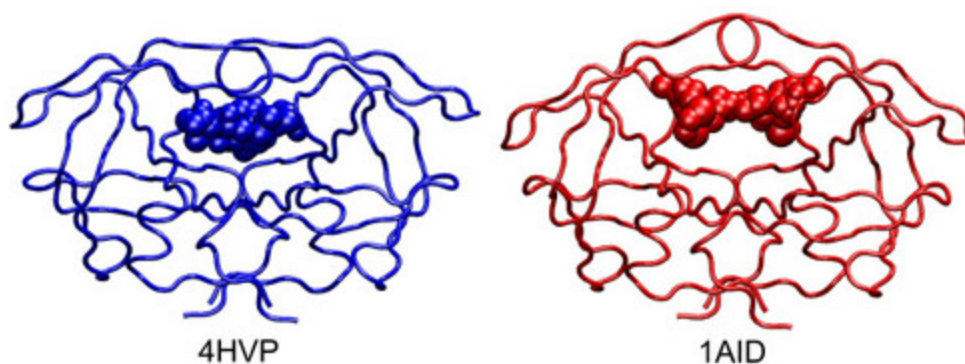


CV-N protein and PCA projections. **a)** A cartoon rendering of the CV-N protein. **b)** The projection of each simulated conformation onto the first two PCs. The concentrated cluster to the left corresponds to the "folded" protein, and the more spread cluster to the right corresponds to the "unfolded" protein. **c)** The PCA coordinates used as a basis for a free-energy (probability) plot.

Figure 4 b) shows the projection of each simulated conformation onto the first two principal components. Each point in the plot corresponds to exactly one of the input conformations, which have been classified along the first two PCs. Note that even the first PC alone can be used to approximately classify a conformation as being "folded" or "unfolded", and the second PC explains more of the data variability by assigning a wider spread to the unfolded region (for which the coordinate variability is much larger). The low-dimensional projections can be used as a basis to compute probability and free-energy plots, for example as in [Das et al.](#) Figure 4 c) shows such a plot for CV-N, which makes the identification of clusters easier. Using PCA to interpolate conformations along the PCs is not a good idea for protein folding data since all of the protein experiences large, non-linear motions of its atoms. Using only a few PCs would quickly destroy the protein topology when large-scale motions occur. In some cases, however, when only parts of a protein exhibit a small deviation from its

equilibrium shape, the main modes of motion can be exploited effectively as the next example shows.

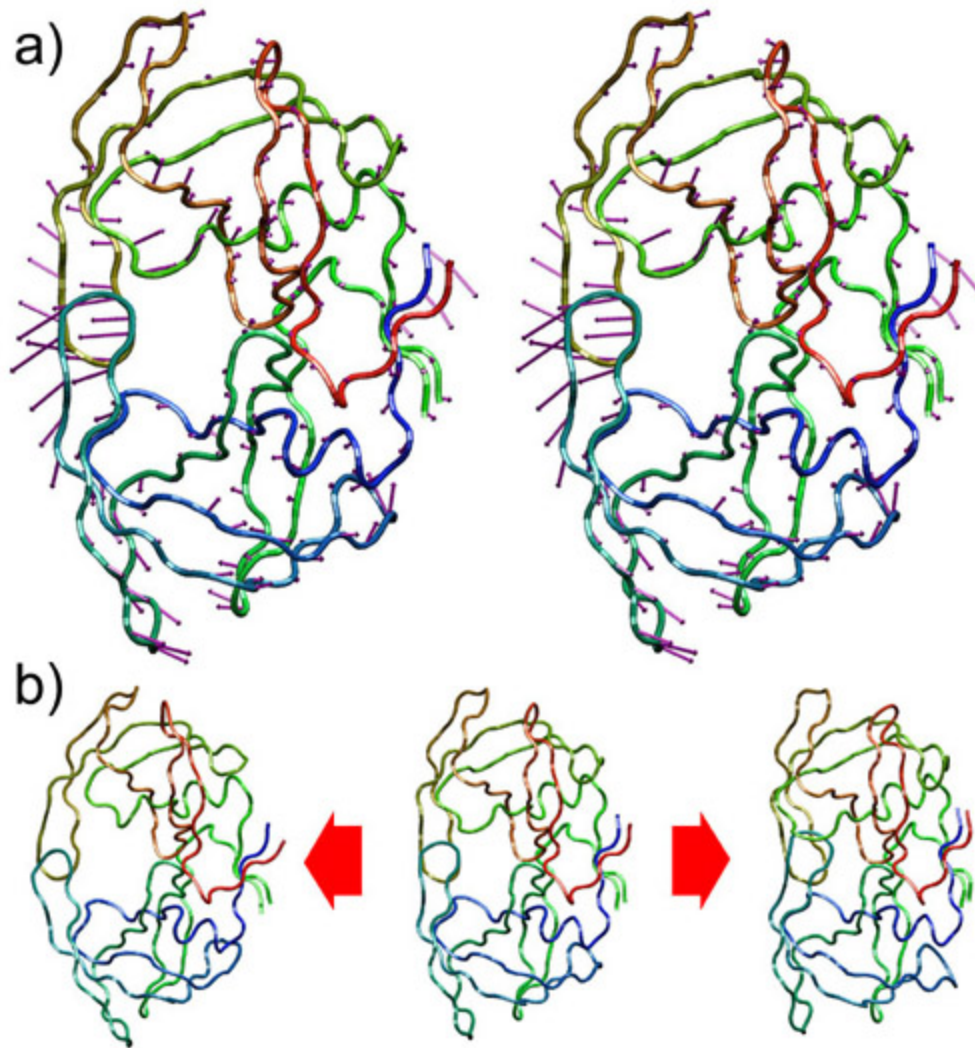
As an example of the application of PCA to analyze the main modes of motion, consider the work of [Teodoro et al.](#), which worked on the HIV-1 protease, depicted in figure 5. The HIV-1 protease plays a vital role in the maturation of the HIV-1 virus by targeting amino acid sequences in the gag and gag-pol polyproteins. Cleavage of these polyproteins produces proteins that contribute to the structure of the virion, RNA packaging, and condensation of the nucleoprotein core. The active site of HIV-1 protease is formed by the homodimer interface and is capped by two identical beta-hairpin loops from each monomer, which are usually referred to as "flaps". The structure of the HIV-1 protease complexed with an inhibitor is shown in figure 5. The active site structure for the bound form is significantly different from the structure of the unbound conformation. In the bound state the flaps adopt a closed conformation acting as clamps on the bound inhibitors or substrates, whereas in the unbound conformation the flaps are more open. Molecular dynamics simulations can be run to sample the flexibility of the flaps and produce an input data set of HIV-1 conformations to which PCA can be applied. A backbone-only representation of HIV-1 has 594 atoms, which amounts to 1,782 degrees of freedom for each conformation.



Alternative conformations for HIV-1 protease. Tube representation of HIV-1 protease (PDB codes 4HVP and 1AID) bound to different inhibitors represented by

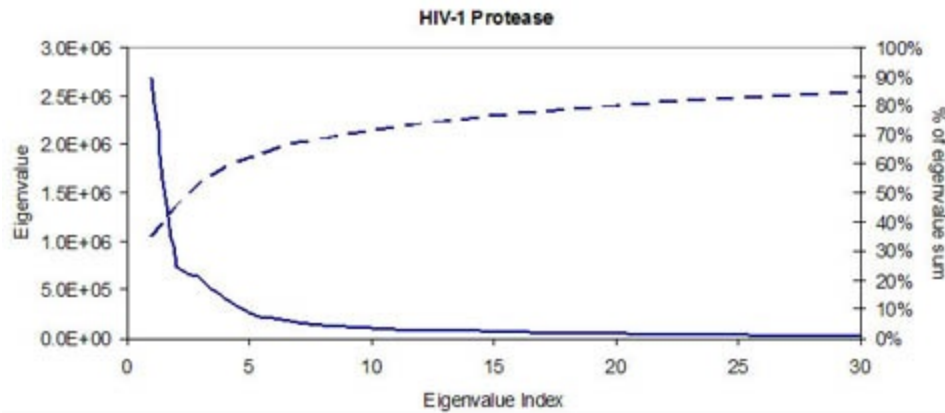
spheres. The plasticity of the binding site of the protein allows the protease to change its shape in order to accommodate ligands with widely different shapes and volumes.

Applying the PCA procedure as outlined before to a set of HIV-1 samples from simulation produces 1,782-dimensional principal components. Since the physical interpretation of the PCs is quite intuitive in this case, the PC coordinates can be split in groups of 3 to obtain the (x,y,z) components for each of the 594 atoms. These components are 3-dimensional vectors that point in the direction each atom would follow along the first PC. In figure 6 a), the per-atom components of the first PC have been superimposed in purple.



First mode of motion for the HIV-1 protease. **a)** The purple arrows are a convenient representation of the first PC grouped every 3 coordinates  $-(x,y,z)$  for each atom to indicate the linear path each atom would follow. Note that the "flaps" have the most important motion component, which is consistent with the simulation data. **b)** A reference structure (middle) can be interpolated along the first PC in a negative direction (left) or a positive one (right). Using only one degree of freedom, the flap motion can be approximated quite accurately.

Figure 6 b) shows the effect of interpolating HIV-1 conformations along the first PC, or first mode of motion. Starting from an aligned conformation from the original data set, multiples of the first PC can be added to produce interpolated conformations. Note that the first mode of motion corresponds mostly to the "opening" and "closing" of the flaps, as can be inferred from the relative magnituded of the first PC components in the flap region. Thus, interpolating in the direction of the first PC produces an approximation of this motion, but using only one degree of freedom. This way, the complex dynamics of the system and the 1,782 apparent degrees of freedom have been approximated by just one, effectively reducing the dimensionality of the representation.



The residual variance (solid line) and percentage of overall variance explained (dashed line) after each principal component.

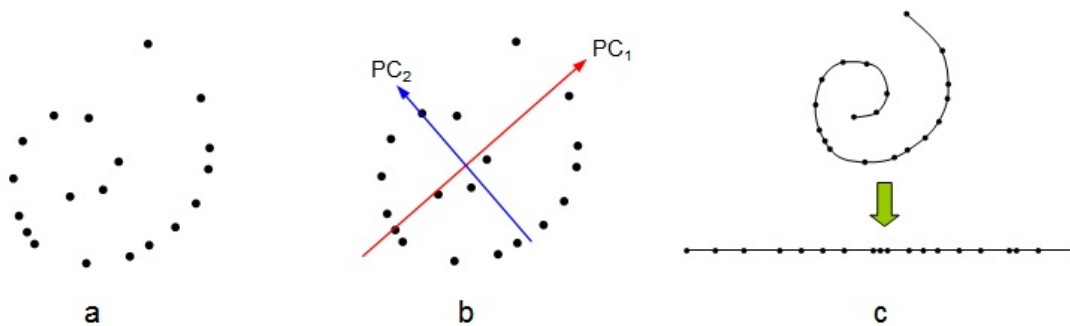
Figure 7 (solid line) shows the residual variance left unexplained by discarding the lower-ranked principal components. Residual variance plots always decrease, and in this case, the first PC accounts for approximately 40% of the total variance along the motion (the dashed line shows the percentage of total variance explained up to the given PC). Also, the first 10 PCs account for more than 70% of the total data variance. Given the

dominance of only a few degrees of freedom it is possible to represent the flexibility of the protein in a drastically reduced search space.

## Non-Linear Methods

PCA falls in the category of what is called a **linear** method, since mathematically, the PCs are computed as a series of linear operations on the input coordinates. Linear methods such as PCA work well only when the collective atom motions are small (or linear), which is hardly the case for most interesting biological processes. Non-linear dimensionality reduction methods do exist, but are normally much more computationally expensive and have other disadvantages as well. However, non-linear methods are much more effective in describing complex processes using much fewer parameters.

As a very simple and abstract example of when non-linear dimensionality reduction would be preferred over a linear method such as PCA, consider the data set depicted in figure 8 a). The data is apparently two-dimensional, and naively considering the data variance in this way leads to two "important" principal components, as figure 8 b) shows. However, the data has been sampled from a one-dimensional, but non-linear, process. Figure 8 c) shows the correct interpretation of this data set as non-linear but one-dimensional.



Effects of dimensionality reduction on an inherently non-linear data

set. **a)** The original data given as a two-dimensional set. **b)** PCA identifies two PCs as contributing significantly to explain the data variance. **c)** However, the inherent topology (connectivity) of the data helps identify the set as being one-dimensional, but non-linear.

Most interesting molecular processes share this characteristic of being low-dimensional but highly non-linear in nature. For example, in the protein folding example depicted in figure 1, the atom positions follow very complicated, curved paths to achieve the folded shape. However, the process can still be thought of as mainly one-dimensional. Linear methods such as PCA would fail to correctly identify collective modes of motion that do not destroy the protein when followed, much like in the simple example of figure 8.

Several non-linear dimensionality reduction techniques exist, that can be classified as either parametric or non-parametric.

- **Parametric methods** need to be given a model to try to fit the data, in the form of a mathematical function called a **kernel**. [Kernel PCA](#) is a variant of PCA that projects the points onto a mathematical hypersurface provided as input together with the points. When using parametric methods, the data is **forced** to lie on the supplied surface, so in general this family of methods do not work well with molecular motion data, for which the non-linearity is unknown.
- **Non-parametric methods** use the data itself in an attempt to infer the non-linearity from it, much like deducing the inherent connectivity in figure 8 c). The most popular methods are Isometric Feature Mapping (Isomap) from [Tenenbaum et al.](#) and Locally Linear Embedding (LLE) from [Roweis et al.](#).

The Isomap algorithm is more intuitive and numerically stable than LLE and will be discussed next.

#### Isometric Feature Mapping (Isomap)

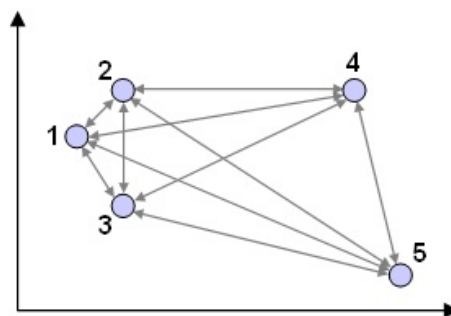
Isomap was introduced by [Tenenbaum et al.](#) in 2000. It is based on an improvement over a previous technique known as MultiDimensional Scaling (MDS). Isomap aims at capturing the non-linearity of a data set from the set itself, by computing relationships between neighboring points. Both MDS and the non-linear improvement will be presented first, and then the Isomap algorithm will be detailed.

**MDS.** Multidimensional Scaling (see [Cox et al.](#)) is a technique that produces a low-dimensional representation for an input set of  $n$  points, where the dimensions are ordered by variance, so it is similar to PCA in this respect. However, MDS does not require **coordinates** as input, but rather, **distances** between points. More precisely, MDS requires as input a data set of points, and a **distance measure**  $d(i,j)$  between any pair of points  $x_i$  and  $x_j$ . MDS works best when the distance measure is a [metric](#). MDS starts by computing all possible pairwise distances between the input points, and placing them in a matrix  $D$  so that  $D_{ij} = d(i,j)$ . The goal of MDS is to produce, for every point, a set of  $n$  Euclidean coordinates such that the Euclidean distance between all pairs match as close as possible the original pairwise distances  $D_{ij}$ , as depicted in figure 9.

Example:  $n=5$  points

Matrix of distances  $D$

	1	2	3	4	5
1	0	1	2	6	7
2	1	0	2	5	6
3	2	2	0	5	6
4	6	5	5	0	3
5	7	6	6	3	0



MDS at work. A matrix of interpoint distances  $D$  is used to compute a set of Euclidean coordinates for each of the input points.

If the distance measure between points has the metric properties as explained above, then  $n$  Euclidean coordinates can always be found for a set of  $n$  points, such that the Euclidean distance between them matches the original distances. In order to obtain these Euclidean coordinates, such coordinates are assumed to have a mean of zero, i.e., they are centered. In this way, the matrix of pairwise distances  $D$  can be converted into a matrix of **dot products** by squaring the distances and performing a "double centering" on them to produce the matrix  $B$  of pairwise dot products:

$$B = -\frac{1}{2}H_n D^2 H_n \quad \text{Where } H_n = \delta_{ij} - \frac{1}{n}$$

Let's assume that  $n$  Euclidean coordinates exist for each data point and that such coordinates can be placed in a matrix  $X$ . Then, multiplying  $X$  by its transpose should equal the matrix  $B$  of dot products computed before. Finally, in order to retrieve the coordinates in the unknown matrix  $X$ , we can perform an SVD of  $B$  which can be expressed as:

$$B = XX^T = Q\Lambda Q^T$$

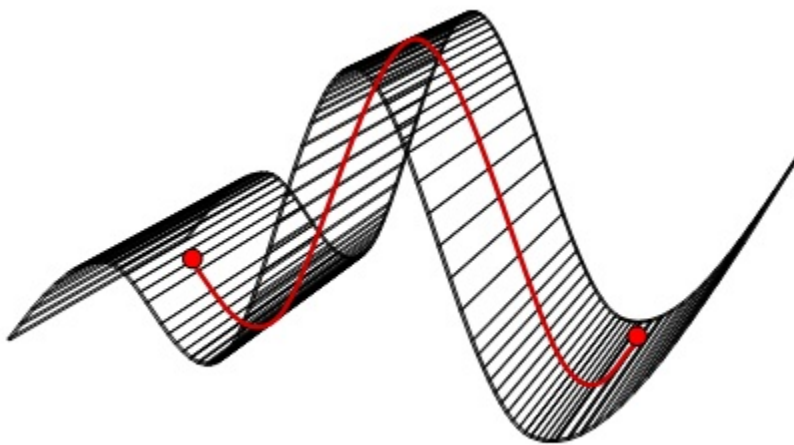
Where the left and right singular vectors coincide because  $B$  is a symmetric matrix. The diagonal matrix of eigenvalues can be split into two identical matrices, each having the square root of the eigenvalues, and by doing this a solution for  $X$  can be found:

$$X = Q\Lambda^{1/2}$$

Since  $X$  has been computed through an SVD, the coordinates are ordered by data variance like in PCA, so the first component of  $X$  is the most important, and so on. Keeping this in mind, the dimensionality reduction can be performed by ignoring the higher dimensions in  $X$  and only keeping the first few, like in PCA.

**Geodesic distances.** The notion of "geodesic" distance was originally defined as the length of a geodesic path, where the geodesic path between

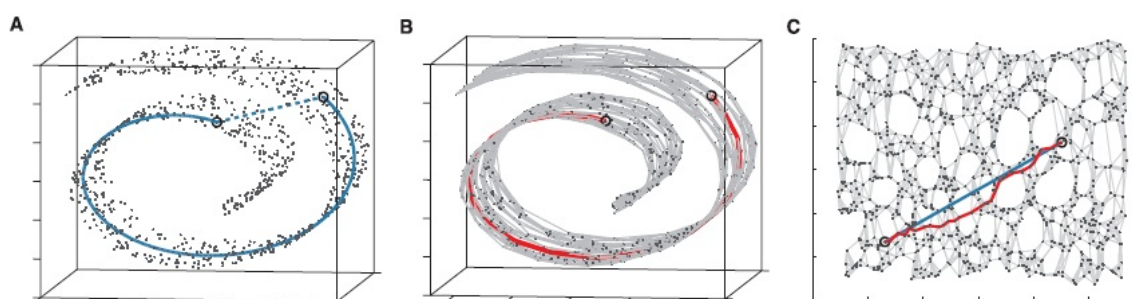
two points on the surface of the Earth is considered the "shortest path" since one is constrained to travel on the Earth's surface. The concept of geodesic distance can be generalized to any mathematical surface, and defined as "the length of the shortest path between two points that lie on a surface, when the path is constrained to lie on the surface." Figure 10 shows a schematic of this generalized concept of geodesic path and distance.



Geodesic distance. The geodesic distance between the two red points is the length of the geodesic path, which is the shortest path between the points, that lies on the surface.

**The Isomap algorithm.** The Isomap method augments classical MDS with the notion of geodesic distance, in order to capture the non-linearity of a data set. To achieve its goal, Isomap uses MDS to compute few Euclidean coordinates that best preserve pairwise **geodesic distances**, rather than **direct distances**. Since the coordinates computed by MDS are Euclidean, these can be plotted on a Cartesian set of axes. The effect of Isomap is similar to "unrolling" the non-linear surface into a natural parameterization, as depicted in figure 11. Isomap approximates the geodesic distances from

the data itself, by first building a **neighborhood graph** for the data. A neighborhood graph consists of the original set of points, together with a connection between "neighboring" points (neighboring points are defined as the closest points to a given point, as determined by the distance measure used) as shown in figure 11-b. After the neighborhood graph has been built, it can be used to approximate the geodesic distance between all pairs of points as the **shortest path** distance along the graph (red path in figure 11-b). Naturally, the sampling of the data set has to be enough to capture the inherent topology of the non-linear space for this approximation to work. MDS then takes these geodesic distances and produces the Euclidean coordinates for the set. Figure 11-c shows two Euclidean coordinates for each point in the example.



Isomap at work on the "swiss roll" data set. **a)** The input data is given as three-dimensional, but is really two-dimensional in nature. **b)** Each point in the data set is connected to its neighbors to form a neighborhood graph, overlaid in light grey. Geodesic distances are approximated by computing shortest paths along the neighborhood graph (red). **c)** MDS applied to the geodesic distances has the effect of "unrolling" the swiss roll into its natural, two-dimensional parameterization. The neighborhood graph has been overlaid for comparison. Now, the Euclidean distances (in blue) approximate the original geodesic distances (in red).

The Isomap method can be put in algorithmic form in the following way:

## The Isomap algorithm

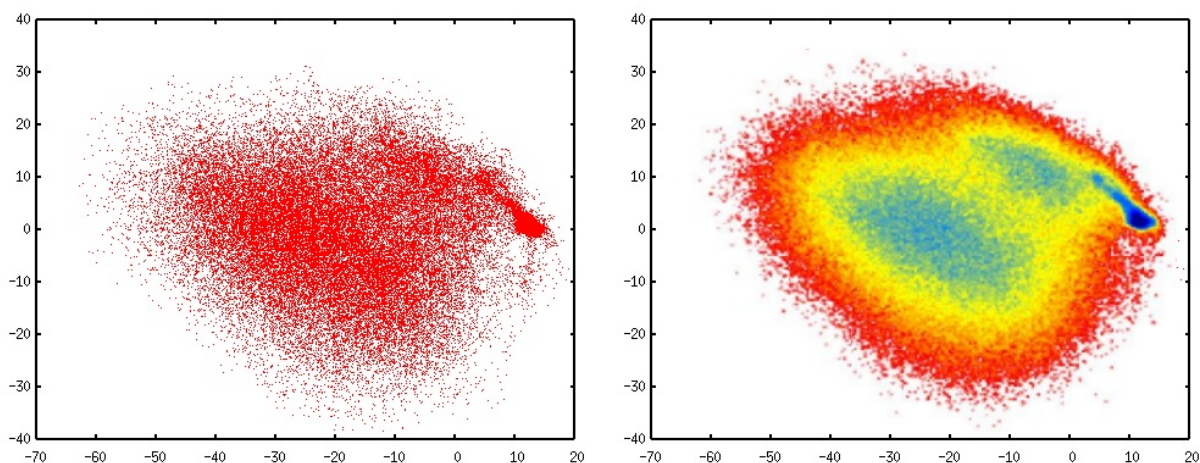
1. **Build the neighborhood graph.** Take all input points and connect each point to the closest ones according to the distance measure used. Different criteria can be used to select the closest neighbors, such as the  $k$  closest or all points within some threshold distance.
2. **Compute all pairwise geodesic distances.** These are computed as the shortest paths on the neighborhood graph. Efficient algorithms for computing all-pairs-shortest-paths exist, such as Dijkstra's algorithm. All geodesic distances can be put in a matrix  $D$ .
3. **Perform MDS on geodesic distances.** Take the matrix  $D$  and apply MDS to it. That is, apply the double-centering formula explained above to produce  $B$ , and compute the low-dimensional coordinates by computing  $B$ 's eigenvectors and eigenvalues.

The Isomap algorithm captures the non-linearity of the data set automatically, from the data itself. It returns a low-dimensional projection for each point; these projections can be used to understand the underlying data distribution better. However, Isomap does not return "modes of motion" like PCA does, along which other points can be interpolated. Also, Isomap is much more expensive than PCA, since building a neighborhood graph and computing all-pairs-shortest-paths can have quadratic complexity on the number of input points. Plus, there is the hidden cost of the distance measure itself, which can also be quite expensive. Regardless of its computational cost, Isomap has proven extremely useful in describing non-linear processes with few parameters. In particular, the work in [Das et al.](#) applied Isomap successfully to the study of a protein folding reaction.

In order to apply Isomap to a set of molecular conformations (gathered, for example, through molecular dynamics simulations) all that is needed is a distance measure between two conformations. The most popular distance measure for conformations is lRMSD, discussed in the module [Molecular Distance Measures](#). There is no need to pre-align all the conformations in this case, since the lRMSD distance already includes pairwise alignment. Thus, the Isomap algorithm as described above can be directly applied to molecular conformations. Choosing an appropriate value for a neighborhood parameter (such as  $k$ ) may require a bit of experience,

though, and it may depend on the amount and variability of the data. It should be noted that now, the shape of the low-dimensional surface where we hypothesize the points lie is unknown to us. In the swiss roll example above, it was obvious that the data was sampled from a two-dimensional surface. For molecular data, however, we do not know, a priori, what the surface looks like. But we know that the process should be low-dimensional and highly non-linear in nature. Of course, the distance measure used as an underlying operation has an impact on the final coordinates as well. The validation of the Isomap method using the lRMSD distance on protein folding data was done in [Das et al.](#), where a statistical analysis method was used to prove its usefulness.

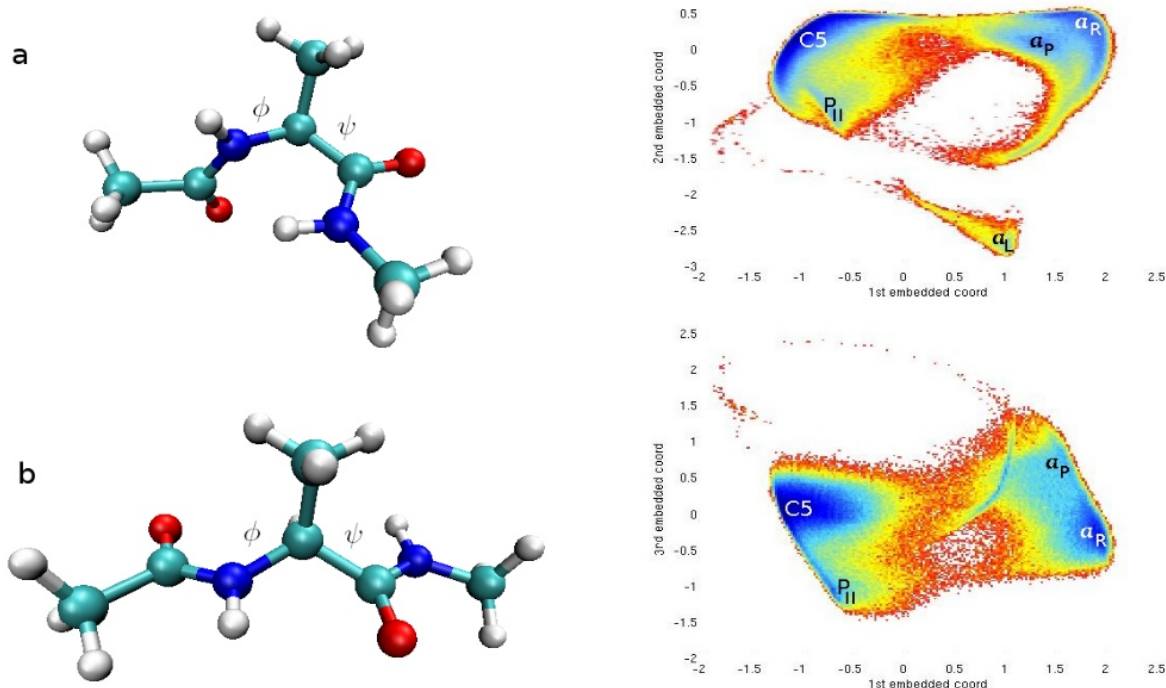
Figure 12 shows the results of applying Isomap to the same molecular trajectory used for the CV-N example in the PCA section. CV-N is a protein that is known to have an "intermediate" state in the folding mechanism, and it is also known to fold through a very ordered process following a well-defined "folding route".



The first two Isomap coordinates for the CV-N protein folding trajectory. **Left:** the projection of each simulated point onto the first two low-dimensional coordinates. **Right:** A free-energy (probability) surface computed using the Isomap coordinates.

The free-energy plot in figure 12 (right) clearly shows the superior quality of the Isomap coordinates when compared with PCA. Isomap clearly identifies the CV-N intermediate (seen as a free-energy minimum in blue) in between the unfolded and folded states. Also, the highest probability route connecting the intermediate to the folded state is clearly seen in this rendering. The PCA coordinates did not identify an intermediate or a folding route, since the PCA coordinates are simply a linear projection of the original coordinates. Isomap, on the contrary, finds the underlying connectivity of the data and always returns coordinates that clearly show the progression of the reaction along its different stages.

The Isomap procedure can be applied to different molecular models. As another, smaller example, consider the Alanine Dipeptide molecule shown in figure 13 (left). This is an all-atom molecule that has been studied thoroughly and is known to have two predominant shapes: an "extended" shape (also called C5 and a variant called PII) and a "helical" shape (also called alpha, with three variants: right-handed, P-like, and left-handed). The application of the Isomap algorithm to this molecule, without any bias by a priori known information, yields the low-dimensional coordinates on the left of figure 13 (shown again as a free-energy plot). The first two coordinates (top right) are already enough to identify all five major states of the peptide, and the possible transitions between them. Applying PCA to such a simple molecule yields comparable results (not shown), but PCA cannot differentiate the left-handed helix shape from the right-handed one. Only the geodesic formulation of Isomap can.



Alanine Dipeptide. Left: **a)** A helical conformation. **b)** An extended conformation. Right: **Top:** free-energy as a function of the first two Isomap coordinates computed from a Molecular Dynamics trajectory. **Bottom:** free-energy as a function of the first and third coordinates, for comparison (these explain the data variance better but do not add new states or transitions).

Although the Isomap coordinates describe non-linear molecular processes extremely well, their computation is very expensive, as mentioned earlier. The most expensive step is the computation of the neighborhood graph. Some approximations are possible to speed up its computation, at the expense of some accuracy in the identification of the "true" nearest neighbors for each point. The work in [Plaku et al.](#) proposed the use of an approximation technique that allows the computation of nearest-neighbors for high-dimensional data sets orders of magnitude faster, by quickly reducing the dimensionality of all points, and then querying for neighbors. The tradeoff between precision and speed is controllable through several algorithm parameters.

## Robotic Path Planning and Protein Modeling

### Topics in this Module

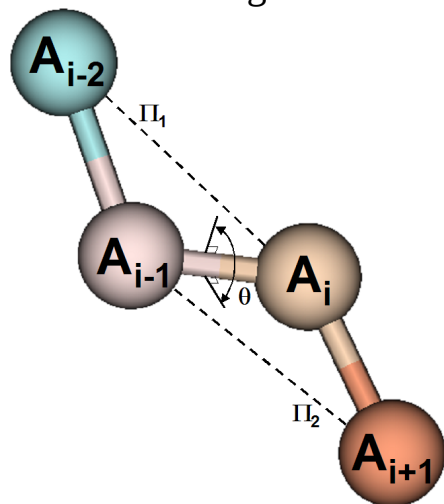
- [Proteins as Robotic Manipulators](#)
- [Robotic Path Planning](#)
- [Sampling-Based Planners for Proteins](#)

### Proteins as Robotic Manipulators

In the modules on [protein kinematics](#) and [inverse kinematics](#), it was shown that, structurally and kinematically, proteins are very similar to a class of **robotic manipulators** consisting of arms connected by revolute joints. Because of this analogy, in the late 1990s, some robotics researchers began to speculate that methods developed for use with robots might also be applicable to the study of proteins. For the remainder of this module, the analogy between robots and proteins will be explored, and then a class of robotics algorithms, called path planners, that will be adapted in a later module to use with proteins, will be introduced.

Recall that a protein is a chain of amino acid residues. Each residue contributes two rotatable dihedral angles, designated  $\phi$  and  $\psi$ , to the main chain of the protein, and may additionally have a side chain with up to five rotatable dihedral angles. Under the **rigid geometry** simplification, these rotatable dihedral angles are the only degrees of freedom available to the protein.

A Dihedral Angle



$\pi_1$  is the plane uniquely defined by the first three atoms  $A_{i-2}$ ,  $A_{i-1}$ , and  $A_i$ . Similarly,  $\pi_2$  is the plane uniquely defined by the last three atoms  $A_{i-1}$ , and  $A_i$ , and  $A_{i+1}$ . The dihedral angle,  $\theta$ , is defined as the smallest angle between these two planes. You can read more about the [angle between two intersecting planes](#).

If we replace each bond by a rigid bar and each rotatable dihedral by a revolute joint, we can build a robotic linkage kinematically equivalent to a protein under rigid geometry.

## Robotic Path Planning

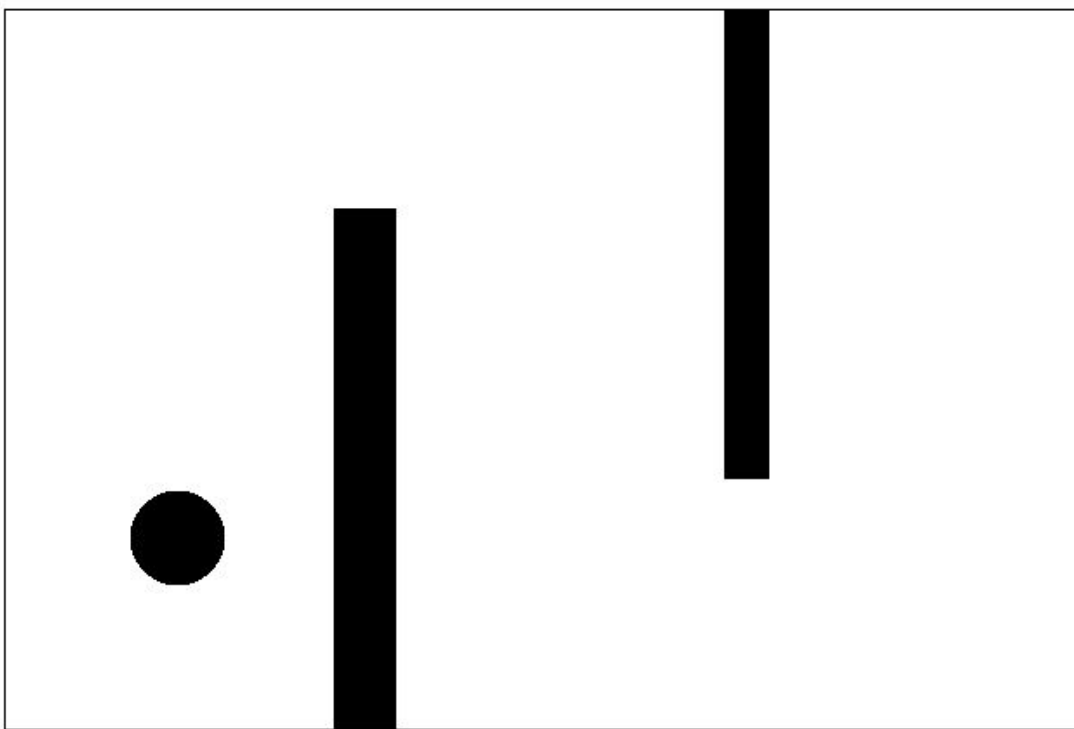
### Background

#### Terms from robotic path planning

- **Work space:** The work space is the geometric space in which a robot operates. It consists of obstacles and empty space that may be occupied by the robot.
- **Configuration:** A configuration of the robot is a full description of the robot's state, including its position, orientation, and the states of any internal degrees of freedom (such as revolute joint angles).

- **Collision:** A configuration is said to be in collision if any part of the robot overlaps with either another part of the robot or with a work space obstacle.
- **Free:** A configuration is said to be free if it is not in collision.
- **Configuration space (C-space):** The space of all configurations of the robot, annotated by whether the robot is in collision or free at each configuration.
- **Free space:** The space of all free configurations.

### Work Space, Two-Dimensional Disc Robot



The work space for a simple robot. The robot is a two dimensional disc. Configurations are therefore  $(x,y)$  pairs. If the robot were not circular, an orientational degree of freedom would also need to be specified.

### Configuration Space, Two-Dimensional Disc Robot



The configuration space corresponding to the above work space. Each point now corresponds to a configuration of the robot, rather than a real point in space. To see how it was constructed, choose any point on the robot as a reference. Now imagine the curves swept out by that point as the robot slides along the surface of the obstacles and along the border of the work space. Every configuration closer to the obstacles than this curve is in collision, and every configuration farther than the curve is free.

In the above figures, the configuration space is two dimensional because the robot has two degrees of freedom. If the heading of the robot mattered (i.e., if the robot were not circular), then a configuration would consist of a position and an orientation. The configuration space would therefore be three dimensional. If the robot had a rotatable joint, this would add another degree of freedom and another dimension to the C-space.

## The Path Planning Problem

The robotic path planning problem is, given a robot, a work space, and starting and goal configurations for the robot in the work space, find a collision-free path for the robot from the starting configuration to the goal, if one exists. Otherwise determine that no such path exists. An extensive introduction to the path planning problem and existing solutions may be found in [\[link\]](#).

Early approaches to path planning included:

- Construction of visibility graphs between the vertices of C-space obstacles.
- Decomposition of the C-space, effectively into subproblems.
- Potential field methods, in which the goal exerts an attractive force on the robot, and the obstacles exert repulsive forces.

The first two methods scale poorly with the dimensionality of the C-space, since the complexity of the C-space affects their run time. Potential fields are subject to local minima. A robot moving down the potential gradient might get stuck in a potential well before it reaches the global potential minimum at the goal.

## Sampling-Based Path Planning

One solution to the scalability problem was to find methods whose run time does not depend on the dimensionality of the C-space, but on some other factor. This led to **sampling-based** path planning. Rather than making some explicit analysis of the whole C-space, sampling based planners built their representation of C-space by sampling random configurations and using a fast collision checker to determine whether they are in collision.

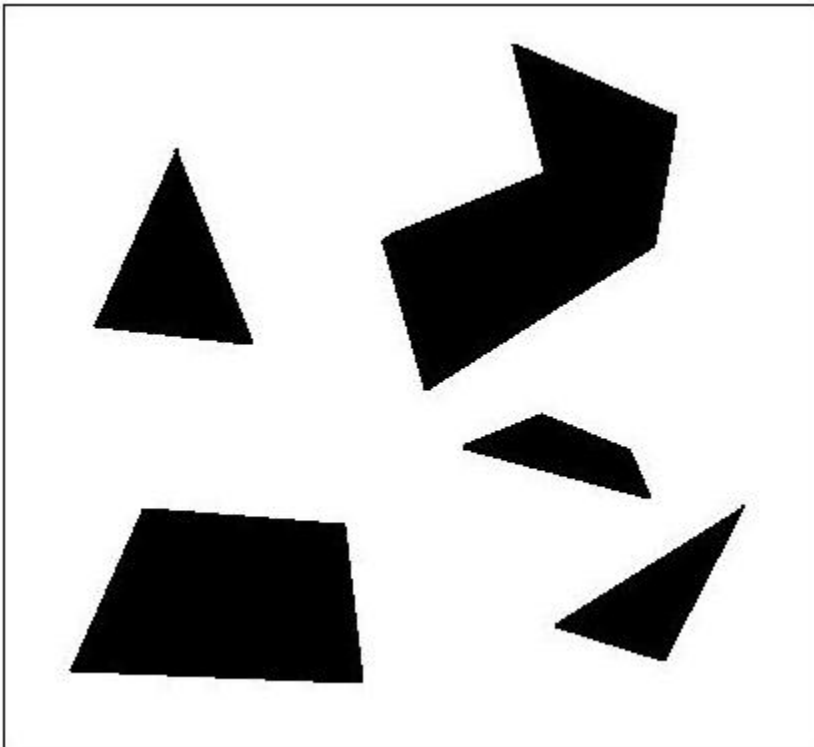
The basis of many modern sampling-based planners is the **Probabilistic Roadmap Method (PRM)** [\[link\]](#). Although the implementation details can become complicated, the basic algorithmic framework is quite straightforward and easy to understand.

## The PRM algorithmic framework:

- Randomly sample a large number of points in C-space, keeping any that are not in collision. This creates a point set in C-space.
- Using a **local planner**, attempt to connect pairs of samples that are relatively close to each other by thoroughly sampling and collision checking configurations between them. This creates a graph data structure called a roadmap.
- To query the roadmap, first attempt to connect the start and goal configurations to the existing graph. If that is successful, search the graph for a path from start to goal using any standard graph search method (often A\*).

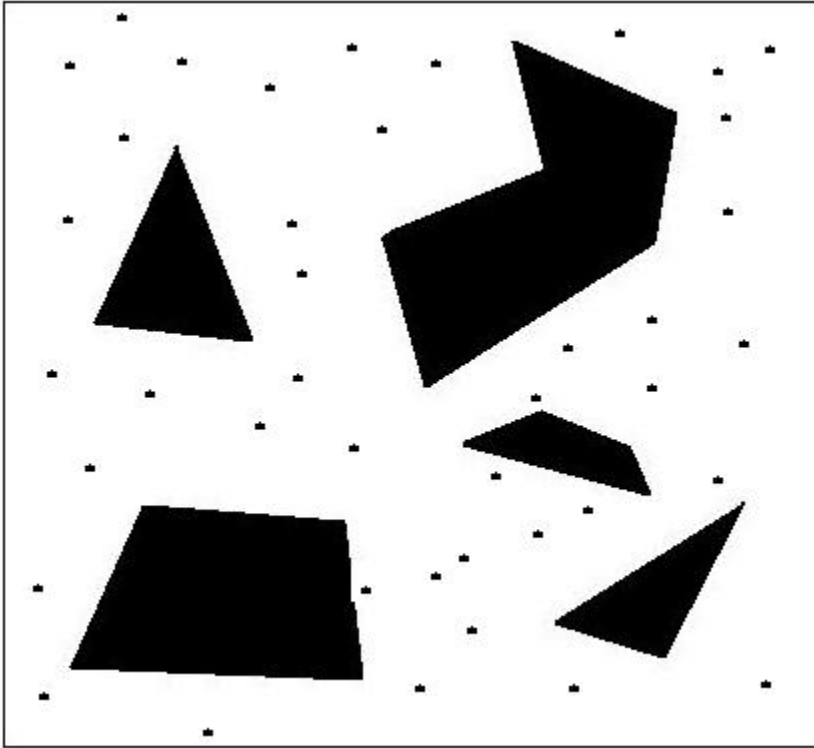
PRM implementations vary in terms of how the points are sampled--remember that random does not mean uniformly at random--as well as in how the local planner attempts to connect nearby configurations.

### Configuration Space for Path Planning



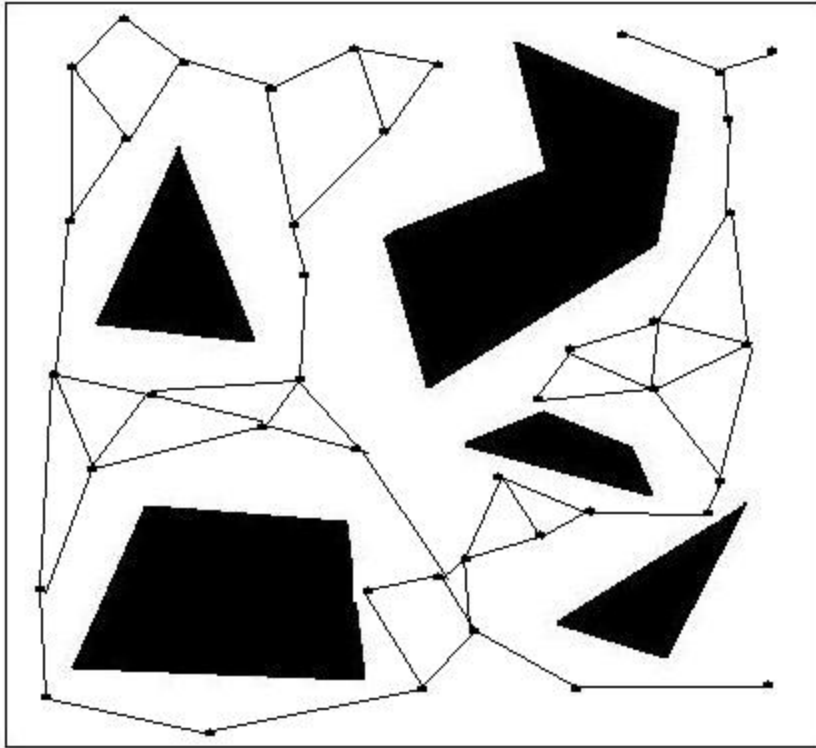
A configuration space in which we wish to build a Probabilistic Roadmap.

## Sampling



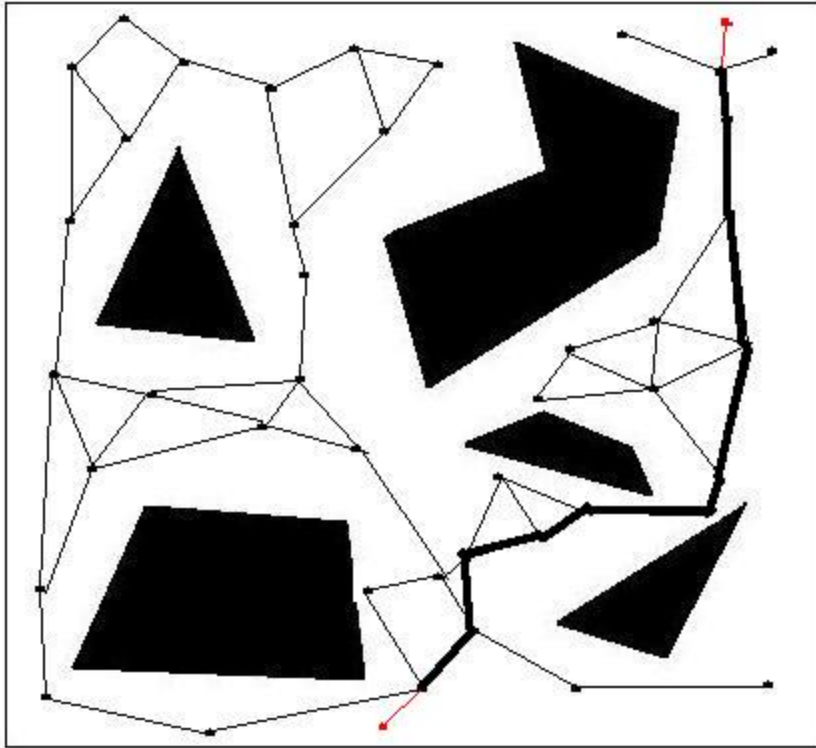
Random points in the configuration space are tested for collision with obstacles. Collision-free points are kept.

## Connection



The roadmap is constructed by connecting nearby samples. Many collision checks are made along each edge to ensure that the connection is legitimate.

Query



The start and goal configurations are connected to their nearest neighbors in the roadmap. A graph search is then made to find the shortest path in the roadmap from start to goal.

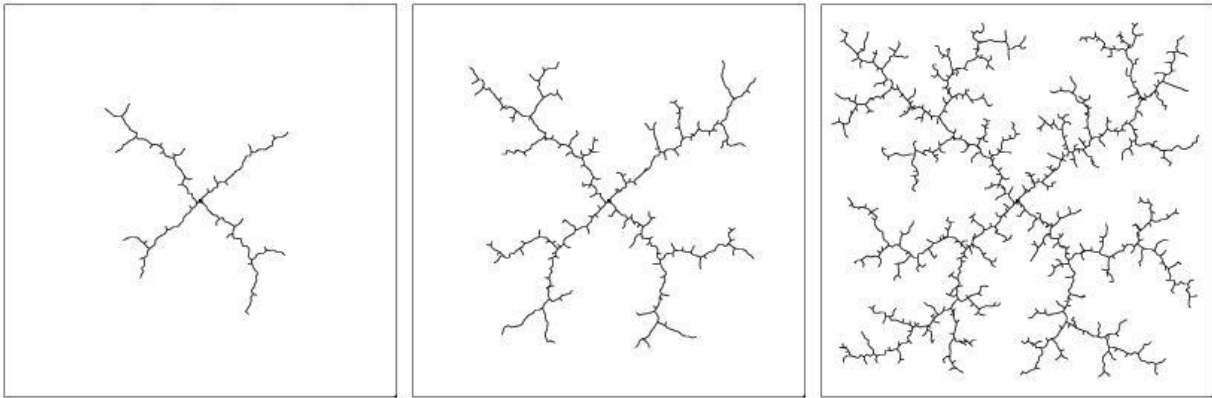
Unlike some slower methods, there is no guarantee that PRM will find a path if one exists. A different kind of guarantee is possible, however. While not complete, PRM is **probabilistically complete**. As the number of samples increases, the probability of the planner finding a path if one exists approaches 1. For many real-world path-planning problems, the method is very fast and reliable in practice.

PRMS are not the only kind of robotic path planner. Building a roadmap is a time-consuming process. The advantage of doing so is that once the roadmap is built, and assuming that the obstacles are not allowed to move, it can be used to rapidly plan an arbitrary number of paths. If the goal is only to find a single path, however, much of the effort of building the

roadmap is wasted. It would be preferable to use a method that is concerned with connecting the start and goal configurations rather than covering the configuration space.

Such a class of sampling-based planners exist, and they are called **tree-based** or **single-query** planners. All of these planners take the same overall approach: They begin with the start configuration for the path planning query, and build a tree data structure of samples growing away from it, with a bias toward the goal configuration.

#### Operation of a Tree-Based Planner



Tree-based planners start at a point and grow outwards, covering the configuration space more and more densely as time goes on.

Single-query planners come in three basic types, each of which has been subject to numerous variations and enhancements since its original development:

#### Types of Single-Query Planners

- **Expansive Spaces Trees (ESTs):** At each step of building an EST, a node of the tree is selected for expansion. Nodes in more sparsely sampled parts of the configuration space are more likely to be chosen, and some bias may also be present for nodes closer to the goal. A number of samples are made in the vicinity of this node, and those that can be connected to it are added to the tree.

- **Rapidly-exploring Random Trees (RRTs):** At each step of RRT-building, a random point is selected in the configuration space, with some bias toward the goal configuration. The nearest node in the tree is found, and a path is created from it toward the random point, either for a set distance or until an obstacle is encountered, whichever comes first. The final collision-free configuration on this path is added to the tree.
- **Path-Directed Subdivision Trees (PDSTs):** PDSTs store edges and branch points as their primitive data rather than nodes. It also maintains a cell decomposition of the configuration space and assigns paths to cells. At each step of building the tree, an edge is deterministically selected based on an estimate of how well-sampled its surroundings are (using the cell decomposition), and a random point on the edge is selected for branching. The old edge is divided in two, a new edge is added at the branch point, and the cell decomposition is updated. Thus, the tree expands outward from its starting point, steered toward less well-sampled regions by the cell decomposition. PDSTs are well-suited for robotics problems with realistic and complex physics.

## Sampling Based Planners for Proteins

With adjustments, one can apply sampling-based robotic path planning to study protein motion. First and foremost, the configuration space, which sorts conveniently into occupied and free configurations, must be replaced by the more complicated molecular **conformation space**. Each conformation of a molecule has a **free energy**, which depends on chemical and physical interactions between its component atoms and those of any other molecules (such as solvent molecules) that may be nearby. This free energy is related to the probability of a protein being in a conformation. Thus, instead of dealing with the binary problem of colliding and free conformations, the conformation space is one of continuously varying probabilities. This problem may be simplified by the use of energy cutoffs, but it is difficult to decide, for any given problem, what energy is so high that a conformation is effectively in collision. Free energy and how it is incorporated into the study of molecular motion is discussed in more detail in [Motion Planning for Proteins: Biophysics and Applications](#).

**Recommended Reading**

Choset, Howie, Kevin M. Lynch, Seth Hutchinson, George Kantor, Wolfram Burgard, Lydia E. Kavraki and Sebastian Thrun. Principles of Robot Motion: Theory, Algorithms, and Implementations. Cambridge, MA: MIT Press, 2005. Chapter 1 for introduction to robotics and analogy between robots and biomolecules. Chapter 7 for a summary of sampling-based planners.

## Motion Planning for Proteins: Biophysics and Applications

### Topics in this Module

- [Free Energy and Potential Functions](#)
  - [Free Energy](#)
  - [Potential Functions](#)
- [Applications of Protein Motion Planners](#)
  - [Kinetics of Protein Folding](#)
  - [Protein-Ligand Docking Pathways and Kinetics](#)

As we suggested in [Robotic Motion Planning and Protein Motion](#), the main difference between modeling a macroscopic robot arm and a protein chain is that the protein is subject to forces resulting from differences in free energy between its states. The protein's **conformation space** does not consist only of colliding and non-colliding structures, but of structures on a continuum of free energy values. In this module, we will provide a very brief overview of free energy as it relates to protein structures, and then give some examples of how path planning techniques have been applied to solving problems in structural biology.

## Free Energy and Potential Functions

### Free Energy

In other modules, we have introduced the concept of a **native conformation** for any given protein, that is, the conformation of the protein that is observed, or expected to be observed, under physiological conditions of temperature, pH, and ion balance. What distinguishes this structure from other structures is that it has the minimum **free energy** of all accessible conformations. There are several different definitions of free energy depending on how the system is defined (for example, whether it is allowed to change in temperature, volume, and/or pressure). One common definition, applicable when temperature and volume are constant, is the Helmholtz Free Energy:

$$F = U - TS$$

Helmholtz free  
energy

The quantity U is the **internal energy** of the system, both kinetic and potential, although for our purposes, we will usually think of changes in U as resulting from changes in potential energy. T is the absolute temperature of the system, and S is the **entropy** of the system, which is very difficult to predict computationally. Entropy is a measure of the number of accessible states to a molecule in a given state, and corresponds to a notion of disorder. In general, the probability of observing a particular state of a system (such as a protein in solution) **increases** exponentially as the free energy **decreases**, in accordance with the Boltzmann distribution:

$$P(E) \propto \exp\left(\frac{-E}{k_B T}\right)$$

Boltzmann-like  
distribution.

E is a particular free energy,  $k_B$  is the Boltzmann constant, and T is the absolute temperature.

In practice, because entropy is very difficult to approximate computationally, potential energy is often used instead of free energy in molecular simulations and docking procedures. When the process is driven by potential energy, this is a reasonable approximation. Some processes are

entropically driven, and results are usually poor when trying to model these processes using only potential energy.

## Potential Functions

**Potential functions** are functions used to evaluate the feasibility of a particular structure of a molecule. Ideally, this would be done with quantum mechanics, in which case the energy function could report the true energy of a particular structure. In practice, quantum mechanical analysis of molecules the size of proteins is wildly intractable. As a compromise, biophysicists have developed artificial functions based on classical physics to approximate the true energy of molecular systems. Sometimes called **potential functions** or **molecular force fields**, these functions generally accept as input a molecular conformation, in the form of Cartesian coordinates for all atoms, and output an energy value. These energy values are generally only meaningful in relative terms: They provide information on what conformations of the molecule are more or less probable than others. The lower the energy value, then the more likely the conformation is to be observed. Most molecular potential functions have the form:

$$E(\vec{R}) = \sum_{\text{bonds}} B(\vec{R}) + \sum_{\text{angles}} A(\vec{R}) + \sum_{\text{torsions}} T(\vec{R}) + \sum_{\text{nonbonded}} N(\vec{R})$$

$\vec{R}$  is the vector representing the conformation of the molecule, typically in cartesian coordinates

A generic potential function.

Approximate energy functions provide the basis for molecular simulations and some protein-ligand docking procedures, among other applications. In some docking problems, a potential function is used to evaluate how likely a particular pose of a small molecule (ligand) in the binding pocket of a protein is. The internal energy of the receptor and the ligand are considered

along with the interaction energy between the two. Interaction energy usually consists of the non-bonded terms found in the internal energy function, summed of all pairs of atoms (r,l), where r is an atom of the receptor and l is an atom of the ligand. If the energy function approximates what is going on well enough then the docked conformation should have minimum energy value. Some docking programs use alternative forms of scoring functions, but in all cases, the object is to find the state of the complex that has the least free energy, and therefore there is a balance between making functions fast to compute and making them reasonably approximate free energy. Potential functions may also be used in simulations to study protein folding mechanisms and kinetics.

## **Terms of energy functions**

### **Bonds**

The bond energy term corresponds to the stretching and compressing of the length of a bond. In most energy functions this term reduces bonds to simple harmonic oscillators, yielding a quadratic equation:

$$E_{\text{bonds}} = K_b(b - b_0)^2$$

where  $K_b$  is an empirically determined constant that depends on the atom types,  $b$  is the current bond's length, and  $b_0$  is the bonds length in equilibrium, which again depends on the atom types. In this case you can think of the bond as a spring, it has an equilibrium length that it wants to remain at. If the bond length varies from the equilibrium length, the energy increases.

### **Bond Angles**

The bond angle energy corresponds to changes in the angle between bonds. As with bond length, the bond angles have an equilibrium value, and any

deviation increases the potential energy. Once again this can be modeled by a simple quadratic term.

$$E_{\text{angles}} = K_{\theta}(\theta - \theta_0)^2$$

where  $K_{\theta}$  is an empirically determined constant,  $\theta$  is the current bond angle, and  $\theta_0$  is the equilibrium angle.

### Torsions

Torsions are created by series of three bonds, and consist of rotations of the bonds on either end with respect to the axis of the middle bond. In molecular structure certain torsional angles are preferred over others and the energy function reflects this. Usually it is described by a Fourier series expansion. The simplest being a single term:

$$E_{\text{tor}} = K_{\text{tor}} [1 + s \times \cos(n\omega)]$$

$K_{\text{tor}}$  is a constant,  $s = +1$  or  $-1$ ,  $n$  is the periodicity and  $\omega$  is the angle

A typical torsional energy term for a potential function.

A more complicated three term expansion can also be used:

$$E_{\text{tor}} = V_1 (1 + \cos(\omega)) + V_2 (1 - \cos(2\omega)) + V_3 (1 + \cos(3\omega))$$

$V_1, V_2, V_3$  are constants, and  $\omega$  is the angle

A more complicated torsional energy term for a potential function.

## Van der Waals Interactions and Steric Clash

Strictly speaking, Van der Waals interactions are weak attractive interactions between atoms at an ideal separation from each other. The atoms transiently induce each other's electron distribution into complementary dipoles, allowing a weak electrostatic attraction between them. In molecular potential fields, Van der Waals attractions are usually combined with steric clash (extremely high energies due to overlapping atoms) in a Lennard-Jones potential, such as this Lennard-Jones 12-6 function:

$$E_{L-J}(i, j) = -4\epsilon_{ij} \left[ \left( \frac{\sigma_{ij}}{r} \right)^{12} - \left( \frac{\sigma_{ij}}{r} \right)^6 \right]$$

where  $\epsilon_{ij}$  is a constant characteristic of the two atom types,  $\sigma_{ij}$  is the average diameter of the two atoms, and  $r$  is the distance between the atom centers.

A typical Lennard-Jones 12-6 potential.

## Electrostatic Interactions

Electrostatic interactions are usually computed using some variant of Coulomb's Law, which assumes that atoms behave as point charges located at their centers. A typical Coulombic term looks like this:

$$E_{\text{elec}} = \frac{q_i q_j}{D r_{ij}}$$

$q_i q_j$  are the charges of the atoms,  $D$  is the effective dielectric constant,  $r_{ij}$  is the distance between the two atoms

Electrostatic energy is computed using a version of Coulomb's Law.

The dielectric constant is a function of the medium through which the two charges interact. The difference between the dielectric constant of water and that of pure protein is substantial, so some models attempt to take it into account. One of the simplest assumes that the farther apart two charges are, the more likely they are to have water between them. This is called a distance-dependent dielectric, because it scales with the distance between the atoms involved.

#### Other Classes of Interactions

While all of the previous terms are almost always included in energy functions, there are a handful of other terms that are common, but not present in every function. These include hydrogen bonding, solvation and cross terms.

Hydrogen bonds (which are not true bonds in the strict, electron-sharing sense) are unusually strong electrostatic interactions, usually between a hydrogen atom and an electronegative atom such as oxygen or nitrogen. They play an important role in determining and maintaining the structure of biomolecules including proteins and nucleic acids. Some energy functions account for hydrogen bonding in the electrostatic term. Other functions include a separate hydrogen bonding term which is most often a Lennard-Jones-like 12-10 potential:

$$E_{\text{hydro}} = K_{ij} \left[ \left( \frac{C_{ij}}{r} \right)^{12} - 2 \left( \frac{D_{ij}}{r} \right)^{10} \right]$$

$K_{ij}$ ,  $C_{ij}$  and  $D_{ij}$  are constants,  $r$  is the distance between the two atoms

A hydrogen-bonding 12-10 term for potential functions.

The solvent that a molecule is in can have a large effect on how it moves. Explicitly representing solvent molecules, however, is a computational cost that most methods try to avoid. Usually the solvent model is separate from the energy function. There are several different ways of approximating solvent interactions including the Generalized Born Model and the Poisson-Boltzmann method. Most force fields do not have an explicit solvent term.

Other terms that describe the interaction between bonds and angles, angles and torsions and so on are included in some force fields. For example to model the interaction between bonds and angles:

$$E_{s-b} = K_{r\theta} (r - r_0) (\theta - \theta_0)$$

$K_{r\theta}$  is a constant,  $\theta_0$  is the equilibrium angle value,  $r_0$  is the equilibrium bond length,  $r$  is the bond length  $\theta$  is the angle value

A potential energy term depending on both bond lengths and angles.

## Parameters

All of the terms presented above include one or more atom-type-dependent constants, or parameters. Determining these parameters is the major problem in developing a new potential function. These parameters are typically found by fitting calculated results to experimental data. Detailed quantum analysis of small molecules may also be used to set some constants. Regardless of how it is determined, it is important to remember that all potential fields are approximations, and most are best suited for some types of proteins over others.

## An Example: The CHARMM All-Atom Empirical Potential

CHARMM (Chemistry at HARvard Macromolecular Mechanics) refers to both a program for macromolecule dynamics and mechanics and the energy function developed for use in that program. CHARMM is a popular force field used mainly for the study of macromolecules. In the most recent version, the parameters were created using experimental data and supplemented with ab initio results. The CHARMM energy function has the form:

$$\begin{aligned} U(\vec{R}) = & \sum_{\text{bonds}} K_b (b - b_0)^2 + \sum_{\text{UB}} K_{UB} (S - S_0)^2 + \sum_{\text{angle}} K_\theta (\theta - \theta_0)^2 \\ & + \sum_{\text{dihedrals}} K_\chi (1 + \cos(n\chi - \delta)) + \sum_{\text{impropers}} K_{\text{imp}} (\phi - \phi_0)^2 \\ & + \sum_{\text{nonbond}} \epsilon \left[ \left( \frac{R_{\text{min}_{ij}}}{r} \right)^{12} - \left( \frac{R_{\text{min}_{ij}}}{r} \right)^6 \right] + \frac{q_i q_j}{\epsilon_1 r_{ij}} \end{aligned}$$

$K_b, K_{UB}, K_\theta, K_\chi, K_{\text{imp}}$  are constants,  $b$  is the bond length,  $b_0$  is the equilibrium bond length,  $S$  is the UB 1,3-distance,  $S_0$  is the ideal UB 1,3-distance,  $\theta$  is the angle value,  $\theta_0$  is the equilibrium angle value,  $\chi$  is the dihedral angle value,  $n$  is the periodicity,  $\phi$  is the improper angle value,  $\phi_0$  is the ideal improper angle value,  $\epsilon$  is the Lennard-Jones well depth,  $R_{\text{min}_{ij}}$  is the distance at the Lennard Jones minimum,  $q_i$  and  $q_j$  are the atoms' charge  $\epsilon_1$  is the effective dielectric constant,  $r_{ij}$  is the distance between the atoms

### The CHARMM all-atom empirical potential function

For more information on CHARMM and the CHARMM force field, please see [The CHARMM website](#).

## Applications of Roadmap Methods

### Kinetics of Protein Folding

The two standard methods of simulating protein motion are molecular dynamics simulation (MD) and Monte Carlo simulation (MC). In MD, a

molecule or system of molecules is given an initial set of atomic momenta, placed in a potential field, and allowed to evolve over time following Newton's equations of motion and the forces exerted on it by the field. In MC, a series of perturbations is applied to a single molecule. After each perturbation, if the estimated energy of the molecule has decreased, the perturbed conformation is used for the next step of the simulation. If the energy has increased, the perturbed conformation might be accepted, with a probability that drops off sharply as the energy change increases. Otherwise, the perturbed conformation is rejected and the previous conformation is perturbed again. Properly implemented MC or MD simulations, run for long enough, should generate a series of conformations with a Boltzmann-like distribution of structures (see the first section of this module for a reminder of what the Boltzmann distribution looks like).

The problem with these methods is that they are very slow. A single MD simulation of a few nanoseconds of motion for an average-size protein, performed on a cluster of processors, can take days, and such simulations are of limited reliability due to approximations of energy and to the extremely short time periods that can be simulated in a reasonable amount of CPU time. Simulations must be repeated to determine what a reasonable, average behavior might look like. Some protein rearrangement events take place on a scale of microseconds, milliseconds, or even seconds, so a trajectory of a few nanoseconds cannot hope to capture these low-frequency events.

The field of chemical kinetics is concerned with the rate at which chemical processes take place, and therefore, the pathways and mechanisms by which they occur. In protein biochemistry, one of the major open questions is the protein folding problem: Given a protein, and its folded (native) and unfolded (denatured) structures, what is the mechanism by which the protein folds into its native state? Currently (2006), it is possible to determine in the laboratory the rate at which a protein folds and sometimes the form of its transition state, the highest energy conformation(s) it assumes in the process of folding. These laboratory measurements can be compared to those inferred from simulation, and the quality of the simulation can thereby be indirectly estimated.

Roadmap-based algorithms to study this problem began with work by Latombe, Singh, and Brutlag in 1999 [\[link\]](#), in which they attempted to use a PRM to find and study protein binding pockets. This work led directly to that of Song and Amato, Apaydin and Latombe, and Singal and Pande, all presented below. Existing methods as of early 2006 are presented.

### A PRM-Based Approach

The research group of Nancy Amato has been working on roadmap-based methods to study the process of protein folding [\[link\]](#)[\[link\]](#)[\[link\]](#)[\[link\]](#). They start with the known native structure of a protein, and incrementally find conformations more and more different from the native state, and build a roadmap using these conformations. The goal is to find a large ensemble of pathways between the native structure and unfolded structures, and to study these pathways and their properties. In their work, the degrees of freedom of the protein are assumed to be the  $\phi$  and  $\psi$  backbone dihedral angles of each amino acid residue. The side chains are assumed to be rigidly attached to the backbone. In their initial work, they generated new conformations for their roadmaps by adjusting the backbone dihedral angles in the folded conformation randomly with various standard deviations. This approach worked well for very small proteins, but did not scale well.

A later sampling approach was based on counting **native contacts**. For the purposes of their method, a native contact was defined as any two alpha-carbons within 7 Å of each other in the folded state of the protein. A new conformation is generated at each step of the sampling phase of the roadmap construction by randomly perturbing some existing sample. The resulting structure was accepted with a probability as follows:

$$P(\text{accept } q) = \begin{cases} 1 & \text{if } E(q) < E_{min}, \\ \frac{E_{max}-E(q)}{E_{max}-E_{min}} & \text{if } E_{min} \leq E(q) \leq E_{max}, \\ 0 & \text{if } E(q) > E_{max}. \end{cases}$$

where  $q$  is the candidate node,  $E(q)$  is the energy of the structure at  $q$ ,  $E_{min}$  is an energy so low that structures with this energy are always accepted, and  $E_{max}$  is an energy so high that all structures with this energy are rejected.

The acceptance criterion for newly sampled conformations.

The energy,  $E$ , used in this research includes a term favoring known secondary structure contacts, and a Lennard-Jones 12-6 term as presented in the previous section. The parameters of the Lennard-Jones term are selected to favor interactions between H and O atoms, and thus hydrogen bonds. The energy thresholds for acceptance are decided by experiment. If accepted, a structure is placed in a bin corresponding to the number of native contacts present, with one bin for each possible number of native contacts from 1 to  $n$ . The procedure begins by sampling around structures in the 100% native contacts bin (initially, just the native structure). Once the next bin, with one fewer contacts, is full, samples are made by perturbing structures in that bin. Thus, the sampling generally proceeds from structures with all native contacts to structures with very few native contacts.

Once the samples are generated, an attempt is made to connect the  $k$  nearest neighbors of each node to the node itself. A series of conformations on the line connecting the two nodes are tested, and if their energy is below a threshold, the edge is included in the roadmap. The weight of the edge depends on the sequence of energies of the conformations computed along the connecting line. The probability of moving from the  $(i-1)$ th structure to the  $i$ th along the line is given by:

$$P_i = \begin{cases} e^{\frac{-\Delta E_i}{k_B T}} & \text{if } \Delta E_i > 0, \\ 1 & \text{if } \Delta E_i \leq 0. \end{cases}$$

where  $P_i$  is the probability of moving from conformation  $i - 1$  to conformation  $i$  along the edge,  $\Delta E_i$  is the difference in energies between conformation  $i - 1$  and conformation  $i$ ,  $k_B$  is the Boltzmann constant, and  $T$  is the absolute temperature.

The weight of an edge is the sum of the logarithms of the probabilities, and is intended to represent the energetic feasibility of making the transition from the conformation represented by one node to the next. This assumes that moving from one node in the roadmap to an adjacent one consists of a series of move along the edge, each associated with a probability. Note that in reality, the path taken by a protein transitioning between two structures need not be a straight line in conformation space.

Once the roadmap is computed, the shortest paths between structures can be found using Djikstra's algorithm, and the folding paths can be extracted and studied. In particular, the order of secondary structure formation can be predicted by a consensus method. The order of secondary structure formation is determined for all paths in the roadmap from unfolded to folded structures, and the most common order is predicted as the true order of formation, which is a coarse, high-level expression of the folding mechanism. On a set of proteins used to test their method, the predicted formation order matched laboratory-determined formation order in all cases where it was available. Because of the coarseness of this notion of the folding mechanism, a statistical analysis of all pathways makes sense.

In their most recent work [\[link\]](#), these researchers have refined the method by which new structures are generated in the sampling phase of roadmap construction. This method is based on **rigidity analysis**. For each structure, each degree of freedom is determined to be independently flexible, dependently flexible, or rigid, using an algorithm called the **Pebble Game** [\[link\]](#). Independently flexible degrees of freedom rotate with no deterministic effect on others. Dependently flexible degrees of freedom force other degrees of freedom to change in a set way. Rigid degrees of freedom are essentially locked in place unless the constraints change. In perturbing an existing structure to generate a new sample, degrees of

freedom are perturbed with a strong bias towards perturbing independently flexible degrees of freedom and against perturbing rigid degrees of freedom. Because the new structures are generated by physically realistic motions, it is expected that they will generally be lower energy and more representative of real structures than if they were generated by completely random perturbation of the degrees of freedom.

In practice, rigidity sampling appears to allow construction of high-quality roadmaps with many fewer samples than were necessary without it. It thus improves the overall efficiency of calculating protein behavior using this roadmap method.

### Stochastic Roadmap Simulations

Numerous variants of MD and MC have been developed in an effort to speed up the process or focus the simulations on particular motions of interest. One method, called the **Stochastic Roadmap Simulation (SRS)** [\[link\]](#) [\[link\]](#) [\[link\]](#) [\[link\]](#) uses a PRM-like structure to approximate a large number of simultaneous MC simulations very rapidly, allowing the analysis of an ensemble of trajectories. This method followed very directly from the first roadmap studies of molecular properties by Singh and Latombe.

The SRS method proceeds as follows:

- N samples are made uniformly at random by selection of a random value for each dihedral angle.
- The k nearest neighbors for each sample are found.
- For each sample, a transition probability is calculated to each of its nearest neighbors, depending on their energy difference as follows:

$$P_{ij} = \begin{cases} \frac{1}{d_j} e^{\frac{-\Delta E_{ij}}{k_B T}} & \text{if } \frac{\epsilon_j/d_j}{\epsilon_i/d_i} < 1, \\ \frac{1}{d_i} & \text{otherwise,} \end{cases}$$

where  $\epsilon_i$  and  $\epsilon_j$  are Boltzmann factors (exponentially proportional to energy) for conformations  $i$  and  $j$ ,  $k_B$  is the Boltzmann constant,  $T$  is the absolute temperature,  $d_i$  and  $d_j$  are the neighbor counts for nodes  $i$  and  $j$ , and  $\Delta E_{ij}$  is the energy difference between structures  $i$  and  $j$ .

## Transition probabilities for SRS.

The energy,  $E$ , in this method is based on the hydrophobic-polar (H-P) energy model, and includes two terms, one favoring hydrophobic interactions, and the other depending on the solvent-excluded volume. Note the difference between the transition probabilities calculated by this method and those calculated by the method presented in the previous section. These probabilities depend only on the energies of the endpoints of an edge, whereas those of the other method depend on the energy along the path between the endpoints. The probabilities of the SRS method are faster to calculate, and, assuming that the system is at equilibrium, more likely to be consistent with the actual distribution of conformations.

- Each sample is given a self-transition probability as follows, so that the sum of outgoing edge probabilities for each node is 1:

$$P_{ii} = 1 - \sum_{i \neq j} P_{ij}$$

where  $P_{ii}$  is the self-transition probability for node  $i$ , and  $P_{ij}$  is the probability of transitioning to state  $j$  if the current state is  $i$ .

Self-transition probabilities ensure that the total transition probability is 1.

The transition probabilities are defined as they are to be consistent with a Boltzmann-like distribution of energies, and therefore with standard Monte Carlo simulation probabilities. The authors demonstrated that each continuous path in the roadmap may be interpreted as a Monte Carlo simulation, and that, if a very large number of samples and edges are made, the aggregate behavior of these Monte Carlo simulations can be analyzed to estimate properties of the protein such as **folding rates** and **transition states**. Essentially, SRS is a way to generate large amounts of Monte Carlo simulation data in a short time. The developers of this method have provided a proof that, for a sufficiently large SRS and a sufficiently long

Monte Carlo simulation, the distribution of conformations is expected to be equal.

To study protein folding using SRS, we observe that some set of nodes in the roadmap represent conformations in or very close to the folded state (native structure). We will refer to this set of nodes as  $F$ . For every node in the roadmap, we can compute an expected number of state transitions (or Monte Carlo steps) to go from that state to a node in  $F$ , with the base case that any node in  $F$  is defined to be at distance 0 from  $F$ . Given a precomputed SRS, we can compute this statistic for each node as follows:

$$t_i = 1 + \sum_{v_j \in \mathcal{F}} P_{ij} + \sum_{v_j \notin \mathcal{F}} P_{ij} t_j, \forall v_i \notin \mathcal{F}$$

where  $t_i$  is the expected number of transitions to get to a node in  $\mathcal{F}$  from node  $v_i$ .  $P_{ij}$  is the probability of transitioning from node  $v_i$  to node  $v_j$  if already at node  $v_i$ .

The expected number of transitions to reach a node in the folded state starting from node  $i$ .

This implies a system of linear equations on the variables  $t_i$ . This system can be solved by an iterative numerical method such as Jacobi iteration. The solution is an estimate for each node of the average number of Monte Carlo steps necessary to achieve a folded conformation.

We can also define a set of nodes representing conformations close to the stable denatured state of the protein as the unfolded state,  $U$ . Given both of the sets  $U$  and  $F$ , we can define a quantity called the transmission coefficient,  $\tau$ , for each node. The transmission coefficient expresses the probability that a structure at a particular node will proceed to a state in  $F$  before it reaches a state in  $U$ --in other words, it is the probability that a given structure will fold before it unfolds. This is often called the folding probability, or  $P_{\text{fold}}$ , in more recent research. The quantity,  $\tau$ , is calculated for each node using the following relation:

$$\tau_i = \sum_{v_j \in \mathcal{F}} P_{ij} \cdot 1 + \sum_{v_j \in \mathcal{U}} P_{ij} \cdot 0 + \sum_{v_j \notin (\mathcal{F} \cup \mathcal{U})} P_{ij} \cdot \tau_j.$$

where  $\tau_i$  is  $P_{fold}$ , the probability that a random walk on the SRS starting from node  $i$  will reach a node in the folded state,  $\mathcal{F}$ , before it reaches a node in the unfolded state,  $\mathcal{U}$ .  $P_{ij}$  is the probability of transitioning to state  $j$  if the current state is  $i$ .

The folding probability for a node  $i$ . This is the probability than a simulation starting at node  $i$  reaches a folded state before reaching an unfolded state.

As before, this relation implies a system of linear equations, this time on  $\tau_i$ , the  $\tau$ -value of each node. Again, it can be solved iteratively, and the result is a  $P_{fold}$  ( $\tau$ ) value for each node.  $P_{fold}$  is an interesting statistic in studying the mechanism of protein folding because structures with a true (as opposed to simulation-derived)  $P_{fold}$  of 0.5 have equal probability of going to the folded or unfolded states, and therefore each one is the highest energy structure on some folding pathway. These are the structures that constitute the **transition state ensemble (TSE)** of the protein, and study of these structures may provide insight into the mechanism by which the protein folds.

## Markovian State Models

**Markovian State Models (MSM)** [\[link\]](#) [\[link\]](#) are roadmaps constructed by running many molecular simulations (Monte Carlo and molecular dynamics) and merging the trajectories. The method starts with a simulation that runs from the folded to unfolded state. It then picks a structure at random (call it  $c$ ) from this trajectory and run a new simulation. If this new simulation reaches the unfolded state, then the next trajectory from which we will pick a structure will consist of the old trajectory from the folded state to  $c$ , and the new trajectory from  $c$  to the unfolded state. If the new trajectory reaches the folded state, we do the opposite. If neither happens in a reasonable time, we reject the new trajectory and start over. This is

repeated a set number of times, and each time a trajectory is accepted, all states from the new part of the trajectory are added to the growing roadmap as nodes, and each transition from the trajectory is added as an edge. When it is added, each edge is labeled with a transition probability of 1 and a transition time equal to the timestep of the simulation.

The goal of this method is to move roadmap methods closer to MD and MC simulations, and in particular to incorporate a notion of time, which follows from the use of simulation techniques in the sampling of new conformations.

Once all of the simulations have been run, nodes that are within some cutoff distance of each other by some similarity metric must be merged. To merge two nodes, one node is removed, and its edges are transferred to the other node. If this results in two edges between the same pair of nodes, the transition probabilities and times are defined as follows:

$$P_{ij}^{\text{new}} = P_{ij}^1 + P_{ij}^2, \quad t_{ij}^{\text{new}} = \frac{P_{ij}^1 t_{ij}^1 + P_{ij}^2 t_{ij}^2}{P_{ij}^1 + P_{ij}^2}.$$

where  $P_{ij}$  is the probability of transitioning to state  $j$  if in state  $i$  and  $t_{ij}$  is the expected time to transition from state  $i$  to state  $j$ .

Expressions for new transition probabilities and transition times when merging nodes in constructing a MSM.

Once all merges are complete, the transition probabilities for each edge are normalized to the range  $[0,1]$  such that the sum of all outgoing edge probabilities from a node is 1. Given the roadmap, Pfold values and folding times can be calculated using the edge probabilities and step times. The approach is the same as with SRS: A system of equations is set up, but instead of Pfold, the value of interest is the expected time for a simulation starting at node  $i$  to reach a folded state, called the **mean first passage time (MFPT)**. The system of equations is solved using standard numerical

methods, as with SRS. On tests of a small protein, called tryptophan zipper beta hairpin, or TZ2, the predicted folding rates agreed well with experiment.

An important fact of all roadmap methods that attempt to extrapolate properties of the entire protein folding landscape is that there is inherent sampling error. The energy landscape of a protein is a continuous function, which roadmap methods attempt to approximate through discrete sampling. The researchers who developed the MSM method also developed a method to estimate the error of the folding rates estimated based on MSMs [\[link\]](#). While a complete description is beyond the scope of this module, the details are available in the 2005 paper by Singhal and Pande linked in the Recommended Reading section below. The error analysis allows them to generate a probability distribution for the folding times for each node in the MSM. Useful in its own right because it gives us an idea of how confident we can be in folding times generated by a given MSM, this analysis is especially useful for focusing sampling during the generation of an MSM.

The variance of the distribution of the folding time for each node provides an estimate of the error. If at each stage of simulation instead of choosing a node at random to start the next simulation, we select the node with the greatest contribution to our estimate of the error in folding time, we effectively focus our efforts where they will decrease the error most. In this way, MSMs with less overall error may be generated with using simulations.

## **Protein-Ligand Docking Pathways and Kinetics**

So far, we have looked at applications of roadmap methods that deal with the single-body problem of protein folding. The first use of roadmaps in molecular modeling, however, was to study the two-body system of protein-ligand docking. The docking problem itself is, given a small molecule and a protein, to predict whether they will bind to form a complex, and if so, what will be the geometry and stability (binding affinity) of this complex. This problem is path-independent, and so does not lend itself to motion planning approaches. Roadmaps can be used, however, to study the question of how

a ligand reaches or exits the binding pocket of a protein, what the energy profile of this process looks like, and the rate at which the ligand binds and dissociates.

Typically, in modeling protein-ligand docking with a roadmap, the protein is held rigid and induces a force field in which the ligand is free to rotate, translate, and change conformation. The first work in this area, by Latombe, Singh, and Brutlag [\[link\]](#), led a few years later to the SRS framework. An SRS for ligand docking pathways can be constructed by starting with the ligand in the bound state, and generating samples for its conformation, location, and orientation in, around, and outside the binding pocket of the protein. These paths can then be studied individually to examine features of the binding process, or as an aggregate to get properties such as binding affinity or escape time, which is represented in an SRS by the weight of paths away from the bound state.

To validate their method for studying docking, the developers of SRS showed [\[link\]](#) that the escape times (in Monte Carlo steps) calculated for ligands leaving proteins with particular mutations in their binding sites were consistent with the expected effect of the mutations: Mutations expected to increase the binding affinity led to longer escape times, and mutations expected to decrease the binding affinity led to shorter escape times. They also showed that SRS could be used to distinguish between the binding site of the protein and other pockets on its surface. Ligands had significantly greater estimated escape times from the true binding site than from spurious ones.

Cortes et al. [\[link\]](#) developed a tree-based sampling method for studying protein-ligand docking pathways. The algorithm is based on the dynamic-domain RRT planner (see [Robotic Path Planning and Protein Modeling](#) for an introduction to RRTs and other tree-based motion planners), in which, when sampling a random point toward which to expand, the location of that point is restricted to be within some distance of the existing tree, rather than anywhere in the whole space. The sampling method is based on the geometry of the system being studied: The major factors contributing to the energy of a conformation are reduced to geometric criteria. Hydrogen bonds and hydrophobic interactions are modeled by distance constraints. Steric

clash is handled by treating atoms as hard spheres and performing collision checks using a fast collision checker called BioCD, developed by the same research group. Only structures satisfying all geometric constraints are subjected to an energy minimization procedure. The geometric constraints help ensure that structures to be added to the tree are already fairly low-energy, ensuring that the minimization can be done quickly, and that time is not wasted minimizing unrealistic structures.

This work was applied to studying the **enantioselectivity** of various proteins. **Enantiomers** are molecules that are non-superimposable mirror images of each other. Although they contain the same atom types and connectivity, enantiomers of a given chemical cannot be interconverted without breaking and reforming bonds. Molecules may contain multiple sites where this kind of asymmetry exists, in which case the molecule may exist as a whole family of **diastereomers**. Most biological molecules have at least one asymmetric center, and are therefore said to be **chiral**, and in most cases, only one diastereomer or enantiomer exists in appreciable quantity. The chemistry of a pair of enantiomers is identical **except** when they are interacting with other chiral molecules, in which case it is important that the correct diastereomer is present for the desired interaction.

Enantioselectivity is the ability of a protein to distinguish between the two enantiomers of a molecule. Since proteins are chiral, they exhibit enantioselectivity for enantiomeric ligands. In the tree-based method of Cortes et al, the amount of time it takes their planner to find an unbound state for a ligand turns out to be correlated with the difficulty of maneuvering the ligand into and out of the binding pocket. Thus, computation times for finding a path out of the binding pocket are much less for the preferred enantiomer of the ligand than for the other enantiomer, often by a factor of 10 or more.

## Recommended Reading

- A PRM-Based Approach
  - Amato, N. M. and G. Song. [PDF](#). Using motion planning to study protein folding pathways. Journal of Computational Biology 9:149-168, 2002.

- Amato, N. M., K. A. Dill, and G. Song. [PDF](#). Using motion planning to map protein folding landscapes and analyze folding kinetics of known native structures. *Journal of Computational Biology* 10:239-255, 2003.
  - Thomas, Shawna, Guang Song and Nancy M. Amato. [PDF](#). Protein Folding by Motion Planning. *Physical Biology* 2:S148-S155, 2005.
  - Thomas, Shawna L., Xinyu Tang, Lydia Tapia, and Nancy M. Amato. [PDF](#). Simulating Protein Motions with Rigidity Analysis. *Proceedings of the 2006 ACM International Conference on Research in Computational Molecular Biology (RECOMB)*, pp. 394-409.
- Stochastic Roadmap Simulations
  - Apaydin, M.S., A. P. Singh, D. L. Brutlag and J.-C. Latombe. [PDF](#). Capturing Molecular Energy Landscapes with Probabilistic Conformational Roadmaps. *Proceedings of the 2001 IEEE International Conference on Robotics and Automation*, pp. 932-939.
  - Apaydin, M. S., C.E. Guestrin, C. Varma, D.L. Brutlag, and J.-C. Latombe. [PDF](#). Stochastic roadmap simulation for the study of ligand-protein interactions. *Bioinformatics*, 18(s2):18-26, 2002.
  - Apaydin, M. S., D. L. Brutlag, C. Guestrin, D. Hsu, J.-C. Latombe and C. Varma. [PDF](#). Stochastic roadmap simulation: an efficient representation and algorithm for analyzing molecular motion. *Journal of Computational Biology* 10:257-281, 2003.
  - Chiang, Tsung-Han, Mehmet Serkan Apaydin, Douglas L. Brutlag, David Hsu and Jean-Claude Latombe. [PDF](#). Predicting Experimental Quantities in Protein Folding Kinetics using Stochastic Roadmap Simulation. *Proceedings of the 2006 ACM International Conference on Research in Computational Molecular Biology (RECOMB)*, pp. 410-424.
- Markovian State Models
  - Singhal, N., C. D. Snow and V. S. Pande. [HTML](#). Using path sampling to build better Markovian state models: predicting the

folding rate and mechanism of a tryptophan zipper beta hairpin. Journal of Chemical Physics 121:415-425, 2004.

- Singhal, Nina and Vijay S. Pande. [HTML](#). Error analysis and efficient sampling in Markovian state models for molecular dynamics. Journal of Chemical Physics 123:204909, 2005.

- Docking Pathways and Kinetics

- Cortes, J, T. Simeon, V. Ruiz de Angulo, D. Guieysse, M. Remauld-Simeon and V. Tran. [PDF](#). A Path Planning Approach for Computing Large-Amplitude Motions of Flexible Molecules. Bioinformatics 21(s1): i116-i125, 2005.

## Protein-Ligand Docking, Including Flexible Receptor-Flexible Ligand Docking

- [Background and Motivation](#)
- [Rigid Receptor Docking](#)
- [Flexible Receptor Docking](#)

### Background and Motivation

Many biological processes involve, at some point, the specific binding a protein to some target molecule. The binding might constitute part of a signalling mechanism between cells, it might be part of a mechanical operation such as muscle contraction, or it might mediate a catalytic event, or it might be part of yet another process. One way that drugs can work is **competitive inhibition**: binding to proteins more strongly than their natural binding partners, and thereby interrupting whatever process the protein mediates.

As an example, consider non-steroidal anti-inflammatory drugs (NSAIDs) such as ibuprofen. These drugs act on a class of proteins called cyclooxygenases (COX), which are involved in the synthesis of chemicals called prostaglandins, which in turn cause pain and inflammation. Inhibition of COX can reduce pain, inflammation, and swelling by substantially reducing the amount of prostaglandins that can be produced. NSAIDs generally work by binding to the active site of COX and blocking it (aspirin and other salicylates are an exception--they disable COX by modifying it chemically).

NSAIDs also illustrate one thing that can go wrong with drugs: side effects. There are actually three classes of COX: COX-1, COX-2, and COX-3. Of the three, COX-2 is the one associated with immune responses, inflammation, and abnormal pain. COX-1 is present in all mammalian cells, as some baseline COX activity is normal. Excessive inhibition of COX in humans is associated with stomach ulcers and indigestion. The problem is one of specificity: In many cases, it is sufficient to inhibit only COX-2 to treat pain and inflammation. In fact, there is a class of NSAIDs called

COX-2 inhibitors that do precisely that. In other cases, side effects can be far more severe and dangerous.

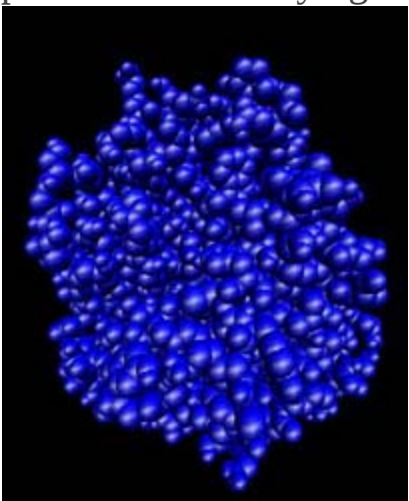
Laboratory techniques for drug discovery are very time-consuming and expensive. Each candidate drug must be synthesized and assayed for activity on the target protein, as well as cross-reactivity with non-targets. There is therefore a great deal of interest in developing computational techniques to assist with this stage of drug development. Although they are still largely an area of research rather than production, a number of automated methods have emerged for identifying promising drug candidates. These methods generally fall into one of two categories:

- **De novo design:** In these approaches, an attempt is made to build a molecule from scratch to fit the binding site of a protein. Often, this involves identifying molecular fragments (often from a database) that are complementary to particular parts of the binding site, and attempting to connect them into a single molecule.
- **Docking:** This approach starts with a database of known molecules and attempts to place each one in the binding pocket of the protein and, if successful, estimates the affinity of the binding using a **scoring function**. In the end, a list of the best-binding molecules for the protein being targeted is returned.

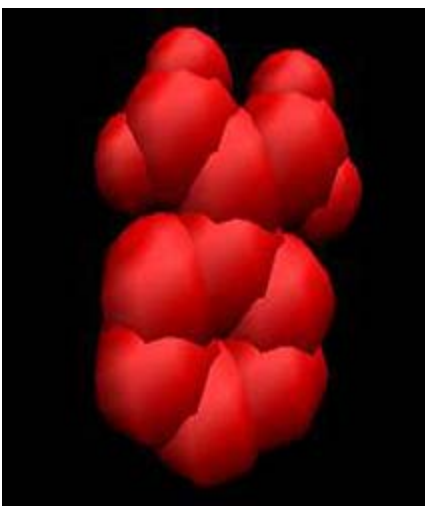
This module is concerned with the latter set of techniques.

Formally, the protein-ligand docking problem is the following: We are given a geometric and chemical description of a protein and an arbitrary small organic molecule. We want to determine computationally whether the small molecule will bind to the protein, and if so, we would like to estimate the geometry of the bound complex, as well as the affinity of the binding. Most algorithms include two components: a search technique to find the optimal placement of the ligand in the binding pocket of the protein, and a scoring function to rate each placement, as well as to rank candidate ligands against each other. The remainder of this module will cover a range of docking approaches, starting with the simplest, rigid-receptor methods, which make very restrictive assumptions about the dynamics of the protein and candidate ligands, and then moving on to more complex approaches

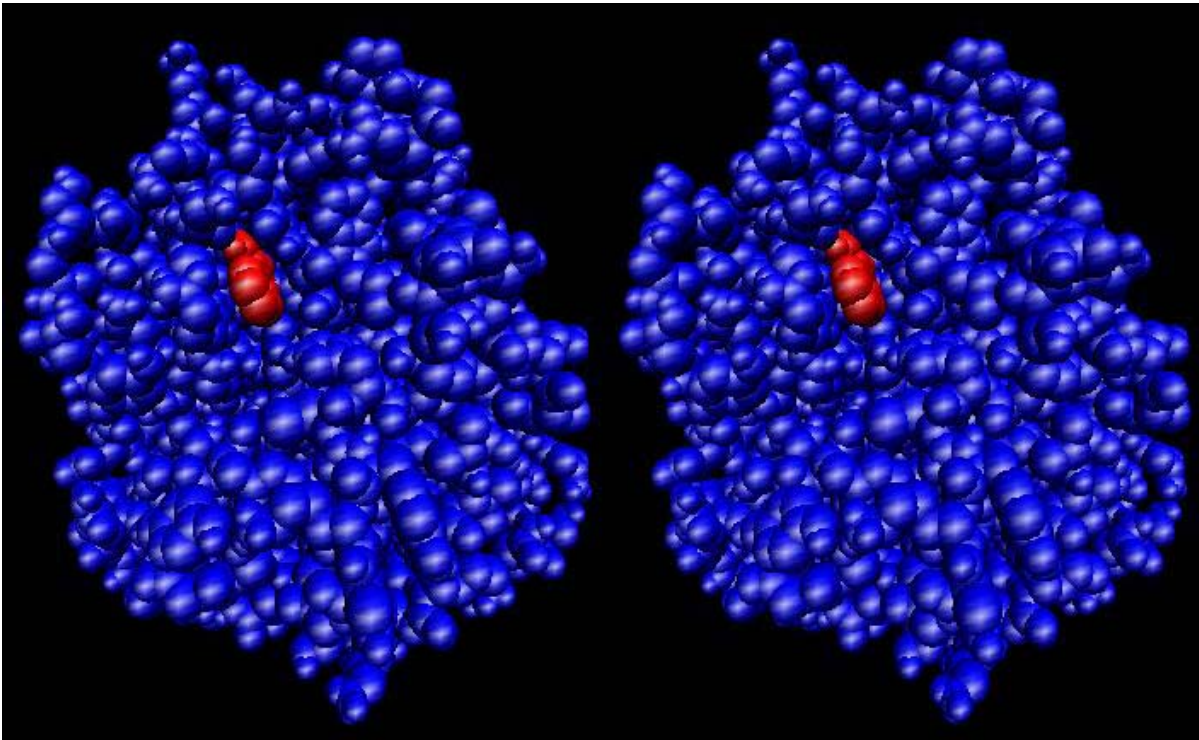
that allow the receptor to change conformation. The latter methods have the potential to identify ligands that might be missed by simpler approaches.



Trypsin, a protease  
involved in  
digestion (PDB  
structure ID 3ptb)



Benzamidine, a  
trypsin inhibitor  
(PDB structure ID  
3ptb)



Stereo view of benzamidinium (red) docked in the active site of trypsin (blue) (PDB structure ID 3ptb)

## **Components of a Docking Program**

As stated earlier, protein-ligand docking methods generally consist of two components: a ligand placement algorithm to enumerate and test possible poses for the ligand in the protein's active site, and a scoring function to evaluate each placement, as well as to evaluate one candidate ligand against another. Each of these components is introduced in more detail below.

### **Ligand placement algorithm**

The first part of any docking technique is a method to place the ligand in various candidate poses in the binding pocket of the receptor. Although

each placement could be completely random and independent, most algorithms either use heuristics based on the chemistry or geometry of the atoms involved (FlexX, DOCK), or use a standard optimization technique such as simulated annealing or a genetic algorithm (Autodock, Gold). A few use explicit molecular dynamics simulation.

## Scoring function

The scoring function provides a way to rank placements of ligands relative to one another. Ideally, the score should correspond directly to the binding affinity of the ligand for the protein, so that the best scoring ligands are the best binders. Scoring functions generally fall into three categories:

### Explicit force field scoring function

Modified versions of both the AMBER and CHARMM force fields (see [this module](#) for more on force fields) have been used as scoring functions. For some complexes, they have been found to provide a good approximation of the free energy of binding. Early versions of Autodock used a subset of the AMBER force field.

### Empirical scoring functions

The score is expressed as a weighted sum:  $\sum_{i \in \text{interactions}} \Delta G_i f_i(l, r)$  where  $\Delta G_i$  is an empirically determined weight for the  $i$ th interaction type. It corresponds to the average free energy contribution of a single interaction of that type over the set of receptor-ligand systems used to normalize the scoring function. The types of interactions that might be included in an empirical scoring function include hydrogen bonds, electrostatic interactions, hydrophobic contacts, solvent exclusion volume, and electrostatic interactions, among others. Examples of empirical scoring functions include the Autodock 3.0 scoring function (see below), Protherics Inc.'s ChemScore [1], and Boehm's SCORE1 [2].

## **Knowledge-based scoring functions**

Knowledge-based scoring functions are derived from a statistical analysis of structures of protein-ligand complexes in the RCSB Protein Databank (PDB). Searches are made for each possible pair of atoms in contact with each other. Interactions found to occur more frequently than would be predicted by random chance are considered attractive (stabilizing), and interactions that occur less frequently are considered repulsive (destabilizing). Examples of knowledge-based scoring functions include Muegge's Potential of Mean Force function [3] and DrugScore [4].

## **Rigid Receptor Docking**

### **Parameterization of the Problem**

Many docking algorithms make the simplifying, but potentially quite inaccurate, assumption that the receptor is a rigid object and attempt to dock the ligand to it. The receptor conformation used is generally one from a receptor-ligand complex whose structure has been determined by x-ray crystallography or NMR spectroscopy. Because the receptor cannot move, the degrees of freedom of the problem are those of the ligand: three translational, three global-rotational, and one internal dihedral rotation for each rotatable bond. It is generally assumed that bond lengths and the angles formed by adjacent bonds do not change, and on the scale of most ligands (10 to 40 atoms), this assumption is a reasonable one. Docking to a rigid receptor is thus an optimization problem over a  $6 + n$  dimensional space, where  $n$  is the number of rotatable bonds in the ligand.

## **Examples of rigid-receptor docking programs**

### **Autodock 3.0**

Link to [The Autodock project home page](#)

Autodock is actually a set of closely related programs and algorithms developed at the Scripps Research Institute and the University of California at San Diego.

#### Search technique

Autodock can use one of several optimization methods to search for the best placement of the ligand:

- Simulated annealing: At each step of simulated annealing, the position and internal rotational state of the ligand is adjusted and the energy calculated. If the energy decreases, the move is accepted. If not, it may be accepted with some probability that depends on the current temperature of the annealing. As the search goes on, the temperature is decreased, and eventually, the final state of the ligand is returned as the docked conformation. Because simulated annealing is a Monte Carlo (randomized) method, different runs will generally produce different solutions.
- A genetic algorithm: The genetic algorithm represents the states of the degrees of freedom of the ligand as a string of digits, and this string is referred to as a gene. A population of different genes is generated at random, and each is scored using the Autodock energy function. Genes are selected to form the next population based on their score, with better scoring genes more likely to be selected. A gene may be selected more than once, and some may not be selected at all. Pairs of the selected genes are allowed to cross over with each other. In this process, a segment of the gene is selected and the values in this range are exchanged between them. The hope is that by combined two partially good solutions, we will eventually find a better solution.
- a Lamarckian genetic algorithm (LGA): This is the same as the standard genetic algorithm except that, before they are scored, each conformation (gene) is subjected to energy minimization. The next population is then founded by members of this energy-minimized population. The name "Lamarckian" refers to the failed genetic theory of Jean-Baptiste Lamarck, who held that an organism could pass on changed experienced in its lifetime to its offspring. This theory was

eventually abandoned in favor of Mendel's now-familiar laws of inheritance. The LGA is faster than both simulated annealing and the standard genetic algorithm, and it allows the docking of ligands with more degrees of freedom.

Autodock uses a kinematic model for the ligand based on rotations around single bonds. The ligand begins the search process from a random location and orientation outside the binding site and by exploring the values for translations, rotations and its internal degrees of freedom, it eventually reaches the bound conformation. Each degree of freedom is encoded as a single gene for the purpose of the genetic algorithm.

The receptor is represented as a potential grid. For each atom type, charge, and placement within the grid, an energy value may be rapidly computed, according to the scoring function below. Precomputation of the grid is time-consuming, but each individual energy calculation is very rapid as a result. The drawback of this approach is that there is no obvious way to introduce protein flexibility. If the were allowed to moves, the entire grid would have to be recomputed at great computational expense.

#### Scoring function

Distinction between docked conformations is carried out by the following empirical scoring function:

$$\Delta G = \Delta G_{VDW} \sum_{i,j} \left( \frac{A_{ij}}{r_{ij}^{12}} - \frac{B_{ij}}{r_{ij}^6} \right) + \Delta G_{hbond} \sum_{i,j} E(t) \left( \frac{C_{ij}}{r_{ij}^{12}} - \frac{D_{ij}}{r_{ij}^{10}} \right) + \Delta G_{elec} \sum_{i,j} \frac{q_i q_j}{\epsilon (r_{ij})^2} + \Delta G_{tor} N_{tor} + \Delta G_{sol} \sum_{i,j} (S_i V_j + S_j V_i) e^{\frac{-r_{ij}^2}{2\sigma^2}}$$

Because the score is an approximation of free energy, lower scores represent greater stability, and the lowest score should correspond to the docked conformation.

## FlexX

Link to [The FlexX home page](#)

FlexX was developed at the Institute for Algorithms and Scientific Computing at the German National Research Center for Computer Science in Sankt Augustin, Germany. The basic procedure is to break the ligand into fragments, then repeatedly place an anchor fragment and incrementally build the entire ligand in place.

### Search technique

For each atom in the ligand and receptor, a set of interaction surfaces is generated and stored. The interaction surfaces represent ideal locations for atoms of the other molecule to form some stabilizing interaction. The shape, size, and location of each surface depends on the type of interaction--hydrogen bonding, electrostatic (ionic), aromatic, or lipophilic (hydrophobic).

The ligand is broken into fragments, separated by rotatable bonds, and a base fragment is chosen. The base fragment is placed by aligning a triangle formed by three of its atoms with interaction surfaces of receptor atoms, using a technique called pose clustering [7]. The choice of base fragment is critical, because a fragment with insufficient interaction surfaces will provide too little guidance for its initial placement. For each sufficiently distinct placement of the base fragment, additional fragments are added in such a way as to maximize interactions and optimize the scoring function.

Because FlexX generates candidate structures by the matching of interaction surfaces, it dramatically decreases the size of the search space compared to a full search of the conformation space, therefore improving the running time. On the other hand, the choice of the anchor fragment is difficult and has the potential to determine which solutions are reachable. In practice, however, FlexX and its derivatives (FlexS, FlexE, and FlexX-Pharm) work well enough to have been incorporated in a number of corporate automated drug discovery applications.

### Scoring function

FlexX uses a variant of the SCORE1 scoring function developed by Hans-Joachim Boehm for the de novo enzyme inhibitor design package LUDI. The scoring function has the following form:

$$\Delta G = \Delta G_0 + \Delta G_{rot} \cdot N_{rot} + \Delta G_{H-bond} \sum_{H-bonds} f(\Delta R, \Delta \alpha) + \Delta G_{ionicinteractions} \sum_{ionicinteractions} f(\Delta R, \Delta \alpha) + \Delta G_{aromatic} \sum_{aromaticinteractions} f(\Delta R, \Delta \alpha) + \Delta G_{lipophilic} \sum_{lipophilicinteractions} f^*(\Delta R)$$

Where  $f$  is a penalty function for deviations from ideal geometry for each kind of interaction, and  $f^*$  is a function penalizing for lipophilic interactions deviating from an ideal separation distance.

### DOCK

#### [The Dock home page](#)

Dock 1.0, first described in 1982 [8], was the first automated receptor-ligand docking program. It was developed in the Department of Pharmacology at the University of California at San Francisco. Dock 4.0, the current version, was released in 1997 [9].

### Search technique

Like FlexX, Dock is driven by the geometry of the ligand and active site. The program approximates the shape of the binding cavity of the receptor with spheres. It then attempts to match the ligand to some subset of the centers of these spheres. Early versions used geometric hashing (see [this module, covering local alignment methods](#)) to perform this matching, but

more recent versions use bipartite graph matching (version 3.5) and single graph matching (version 4.0) for improved speed.

#### Scoring function

Dock offers three scoring functions. The first is based on an approximation to the Lennard-Jones potential (Van der Waals interactions). This essentially enforces geometric alignment and shape constraints. The second uses the program DELPHI to calculate the electrostatic potential of the complex. The third calculates the energy of the complex under the AMBER force field.

#### Links

- [HotDock, a human-guided docking program](#)
- [Bio.com, a clearinghouse of the latest information about the bio-tech industry](#)
- [Babel, a program to convert between most common molecular file formats](#)
- [Software developed by the Theoretical and Computational Biophysics group at UIUC and the Beckman Institute. Includes NAMD, a molecular dynamics simulator, and VMD, a molecule visualizer.](#)
- [Protein Explorer \(Rasmol\), a simple molecule visualizer.](#)

## Flexible Receptor Docking

### Introduction

As previously mentioned, docking entails determining not only the identity and three dimensional structure of the bound ligand, but also how the binding process affects the conformation of the receptor. This section will review the different receptor flexibility representations that have been proposed to study receptor conformational changes in the context of structure based drug design.

A central paradigm which was used in the development of the first docking programs was the lock-and-key model first described by Fischer [10]. In this model the three dimensional structure of the ligand and the receptor complement each other in the same way that a lock complements a key. However, further work confirmed that the lock-and-key model is not the most correct description for ligand binding. A more accurate view of this process was first presented by Koshland [11] in the **induced fit** model. In this model the three dimensional structure of the ligand and the receptor adapt to each other during the binding process. It is important to note that not only the structure of the ligand but also the structure of the receptor changes during the binding process. This occurs because the introduction of a ligand modifies the chemical and structural environment of the receptor. As a result, the unbound protein conformational substates, corresponding to the low energy regions of the protein energy landscape are likely to change. The induced fit model is supported by multiple observations in different proteins such as streptavidin, HIV-1 protease, DHFR, aldose reductase and many others.

More information about some of these proteins and other proteins motions can be found at the following links:

- [HIV Protease Database](#)
- [DHFR the MOVIE homepage](#)
- [Database of Macromolecular Movements](#)

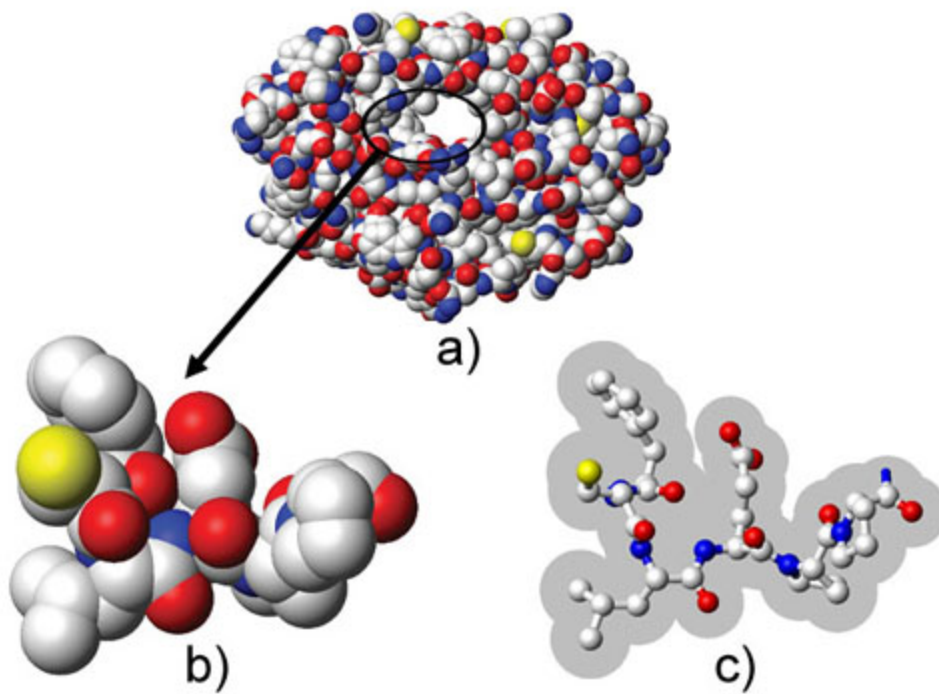
Although it has been clearly established that a protein is able to undergo conformational changes during the binding process, most docking studies consider the protein as a rigid structure. The reason for this crude approximation is the extraordinary increase in computational complexity that is required to include the degrees of freedom of a protein in a modeling study. There is currently no computationally efficient docking method that is able to screen a large database of potential ligands against a target receptor while considering the full flexibility of both ligand and receptor. In order for this process to become efficient, it is necessary to find a representation for protein flexibility that avoids the direct search of a solution space comprised of thousands of degrees of freedom. What follows is a brief review of the different representations that have been used to

incorporate protein flexibility in the modeling of protein/ligand interactions. A common theme behind all these approaches is that the accuracy of the results is usually directly proportional to the computational complexity of the representation. The different types of flexibility representations models are grouped into categories that illustrate some of the key ideas that have been presented in the literature in recent years. However it is important to note that the boundaries between these categories are not rigid and in fact several of the publications referenced below could easily fall in more than one category.

## **Flexibility Representations**

### **Soft Receptors**

Perhaps the simplest solution to represent some degree of receptor flexibility in docking applications is the use of soft receptors. Soft receptors can be easily generated by relaxing the high energy penalty that the system incurs when an atom in the ligand overlaps an atom in the receptor structure. By reducing the van der Waals contributions to the total energy score the receptor is in practice made softer, thus allowing, for example, a larger ligand to fit in a binding site determined experimentally for a smaller molecule (see Figure 6). The rationale behind this approach is that the receptor structure has some inherent flexibility which allows it to adapt to slightly differently shaped ligands by resorting to small variations in the orientation of binding site chains and backbone positions. If the change in the receptor conformation is small enough, it is assumed that the receptor is capable of such a conformational change, given its large number of degrees of freedom, even though the conformational change itself is not modeled explicitly. It is also assumed that the change in protein conformation does not incur a sufficiently high energetic penalty to offset the improved interaction energy between the ligand and the receptor. The main advantage of using soft receptors is ease of implementation (docking algorithms stay unchanged) and speed (the cost of evaluating the scoring function is the same as for the rigid case).



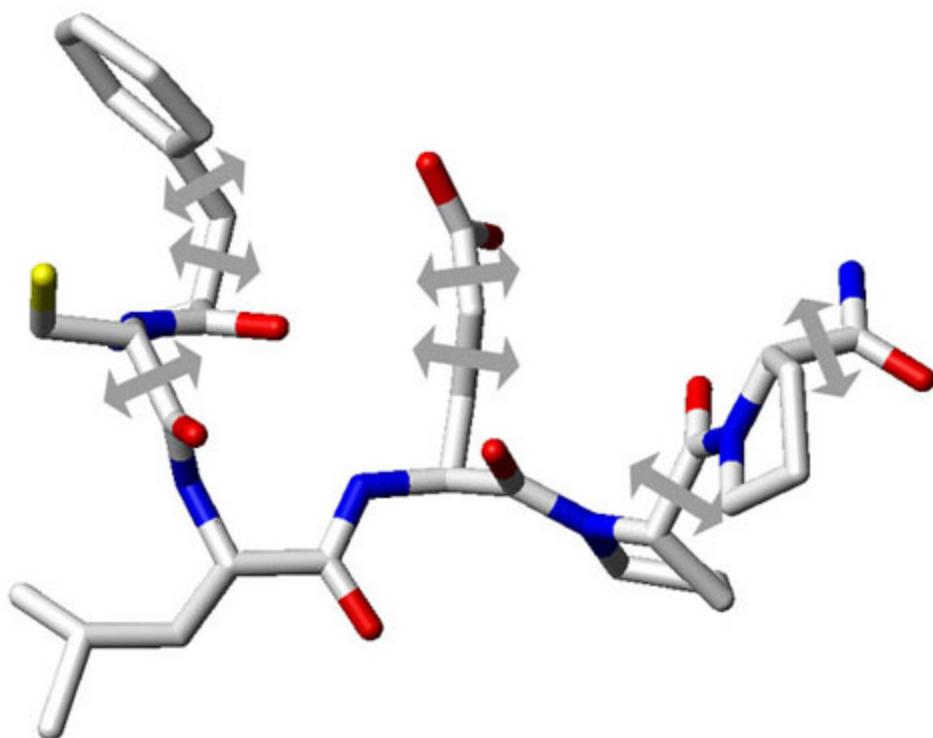
a) Three dimensional van der Waals representation of a target receptor. b) Close up image of a section of the binding site. For the purposes of rigid protein docking, the receptor is commonly described by the union of the volumes occupied by its atoms. The steric collision of any atom of the candidate ligand with the atoms of the receptor will result in a high energetic penalty. c) Same section of the binding site as shown in b) but with reduced radii for the atoms in the receptor. This type of soft representation allows ligand atoms to enter the shaded area without incurring a high energetic penalty.

Another use of soft docking models is to improve convergence during energy minimization of the complex by avoiding local minima. In the initial stages of the conformational search the ligand is allowed to overlap with the receptor and nonbonded energy terms are modified to avoid high energy gradients. During the course of the minimization the interactions are then gradually restored to their original values simulating a ligand that is

gradually exposed to the field of the receptor. This allows initial ligand/receptor conformations, which due to steric clashes would result in a very high energy penalty, to slowly adapt to each other in a complementary conformation without overlaps. One potential pitfall of this approach is the possibility that the ligand may become interlocked with the protein, leading to failure of the docking procedure. Although the use of soft receptors presents a number of advantages such as ease of implementation and computation speed, it also makes use of conformational and energetic assumptions that are difficult to verify. This can easily result in errors, especially if the soft region is made excessively large to account for larger conformational changes on the part of the receptor.

### **Selection of Specific Degrees of Freedom**

In order to reduce the complexity of modeling the very large dimensional space representing the full flexibility of the protein, is it possible to obtain an approximate solution by selecting only a few degrees of freedom to model explicitly. The degrees of freedom chosen usually correspond to rotations around single bonds (see Figure 7). The reason for this choice is that these degrees of freedom are usually considered the natural degrees of freedom in molecules. Rotations around bonds lead to deviations from ideal geometry that result in a small energy penalty when compared to deviations from ideality in bond lengths and bond angles. This assumption is in good agreement with current modeling force fields such as CHARMM [\[12\]](#) and AMBER [\[13\]](#). Selection of which torsional degrees of freedom to model is usually the most difficult part of this method because it requires a considerable amount of a priori knowledge of alternative binding modes for a given receptor. This knowledge usually is a result of the availability of experimental structures obtained under different conditions or using different ligands. If multiple experimental structures are not available some insight can be obtained from simulation methods such as Monte Carlo (MC) or molecular dynamics (MD). The torsions chosen are usually rotations of aminoacid side chains in the binding site of the receptor protein. It is also common to further reduce the search space by using rotamer libraries for the aminoacid side chains



Stick representation of the same binding site section as shown in Figure 1. In order to approximate the flexibility of the receptor it is possible to carefully select a few degrees of freedom. These are usually select torsional angles of sidechains in the binding site that have been determined to be critical in the induced fit effect for a specific receptor. In this example the selected torsional angles are represented by arrows.

An example of a program that takes the approach of selecting a few degrees of freedom to represent protein flexibility is the program GOLD [\[14\]](#). In GOLD, Jones et al. use a genetic algorithm (GA) to dock a flexible ligand to a semi-flexible protein. GAs are an optimization method that derives its behavior from a metaphor of the process of evolution. A solution to a problem is encoded in a chromosome and a fitness score is assigned to it based on the relative merit of the solution. A population of chromosomes then goes through a process of evolution in which only the fittest solutions

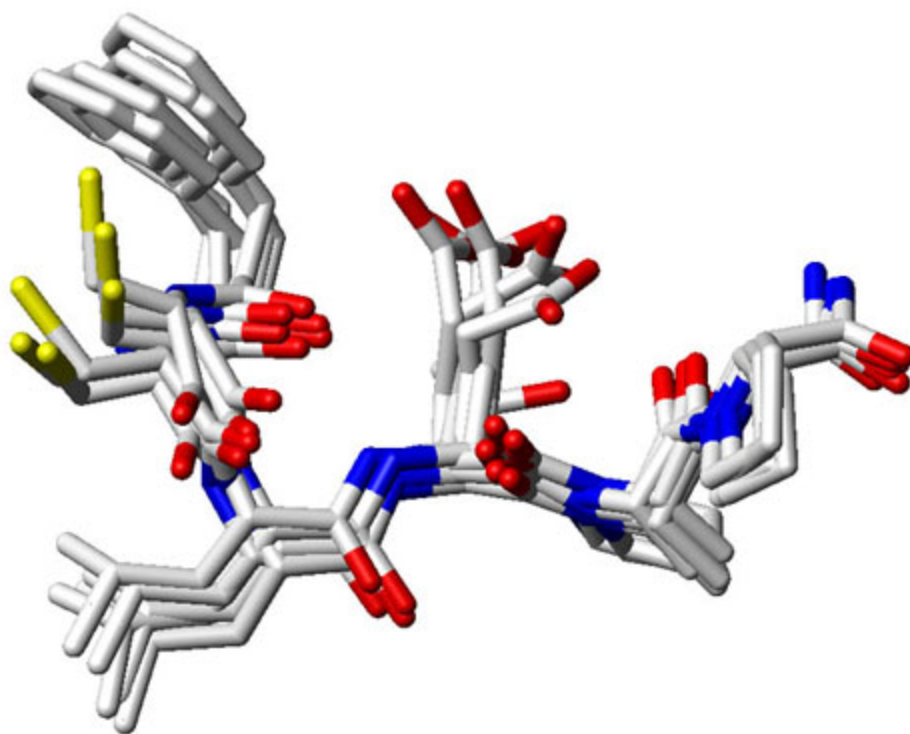
survive. This program takes into account not only the position and conformation of the ligand but also the hydrogen bonding network in the binding site. This was achieved by encoding orientation information for donor hydrogen atoms and acceptors in the GA chromosome. This type of conformational information is very important because if the starting point for a docking study is a rigid crystallographic structure, the orientations of hydroxyl groups will be undetermined. Being able to model these orientations explicitly removes any bias that might result from positioning hydroxyl groups based upon a known ligand. One limitation of this work is that the binding site still remains essentially rigid because protein conformational changes are limited to a few terminal bonds. This program performed very well for hydrophilic ligands but encountered some difficulties when trying to dock hydrophobic ligands due to the reduced contribution of hydrogen bonding to the binding process. More information about GOLD can be found at the following link:  
<http://www.ccdc.cam.ac.uk/prods/gold/>

### **Multiple Receptor Structures**

One possible way to represent a flexible receptor for drug design applications is the use of multiple static receptor structures (see Figure 8). This concept is supported by the currently accepted model that proteins in solution do not exist in a single minimum energy static conformation but are in fact constantly jumping between low energy conformational substates. In this way the best description for a protein structure is that of a conformational ensemble of slightly different protein structures coexisting in a low energy region of the potential energy surface. Moreover the binding process can be thought of as not exactly an induced fit model as described by Koshland in 1958 [11], but more like a selection of a particular substate from the conformational ensemble that best complements the shape of a specific ligand.

The use of multiple static conformations for docking gives rise to two critical questions. The first question is: How can we obtain a representative subset of the conformational ensemble typical of a given receptor? Currently there exist only a limited set of means to generate the three

dimensional structure of macromolecules. The structures can be determined experimentally either from X-ray crystallography or NMR, or generated via computational methods such as Monte Carlo or molecular dynamics simulations. Simulations typically use as a starting point a structure determined by one of the experimental methods. Ideally we would like to use a sampling that provides the most extensive coverage of the structure space. Comparisons done between traditional molecular simulations and experimental techniques [\[15\]](#), [\[16\]](#) seem to indicate that X-ray crystallography and NMR structures seem to provide better coverage. However this balance can potentially change due to advances in computational methods. Another limitation in choosing data sources is availability. Although experimental data is preferable, the monetary and time cost of determining multiple structures experimentally is significantly higher than obtaining the same amount of data computationally. The second critical question is: What is the best way of combining this large amount of structural information for a docking study? This question also remains open. Current approaches use diverse ways of combining multiple structures.



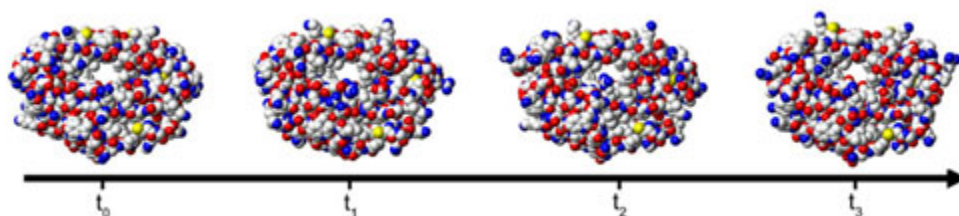
Superposition of multiple conformers of the same binding site section as shown in Figure 1. As an alternative to considering the target protein as a single three dimensional structure, it is possible to combine information from multiple protein conformations in a drug design effort. These can be either considered individually as rigid representatives of the conformational ensemble or can be combined into a single representation that preserves the most relevant structural information.

One of the main advantages of using multiple structures instead of using a selection of degrees of freedom to represent protein flexibility is that the flexible region is not limited to a specific small region of the protein. Multiple structures allow the consideration of the full flexibility of the protein without the exponential blow up in terms of computational cost that would derive from including all the degrees of freedom of the protein. On the other hand, flexibility is modeled implicitly and as such only a small fraction of the conformational space of the receptor is represented. In addition, the method by which the multiple receptor structures are combined has a drastic influence on the possible results of the docking computation.

### **Molecular Simulations**

To simulate the binding process with as much detail as possible and avoid some of the limitations of previous flexibility models one can use force field based atomistic simulation methods such as Monte Carlo or molecular dynamics (see Figure 9). Whereas molecular dynamics applies the laws of classical mechanics to compute the motion of the particles in a molecular system, Monte Carlo methods are so called because they are based on a random sampling of the conformational space. The main advantage of Monte Carlo or molecular dynamics flexibility representations in docking

studies is that they are very accurate and can model explicitly all degrees of freedom of the system including the solvent if necessary. Unfortunately, the high level of accuracy in the modeling process comes with a prohibitive computational cost. For example, in the case of molecular dynamics, state of the art protein simulations can only simulate periods ranging from 10 to 100 ns, even when using large parallel computers or clusters. Given that diffusion and binding of ligands takes place over a longer time span, it is clear that these simulations techniques cannot be used as a general method to screen large databases of compounds in the near future. It is however possible to carry out approximations that reduce the computational expense and lead to insights that would be impossible to gain using less flexible receptor representations. The cost of carrying out the computational approximations is usually a loss in accuracy.

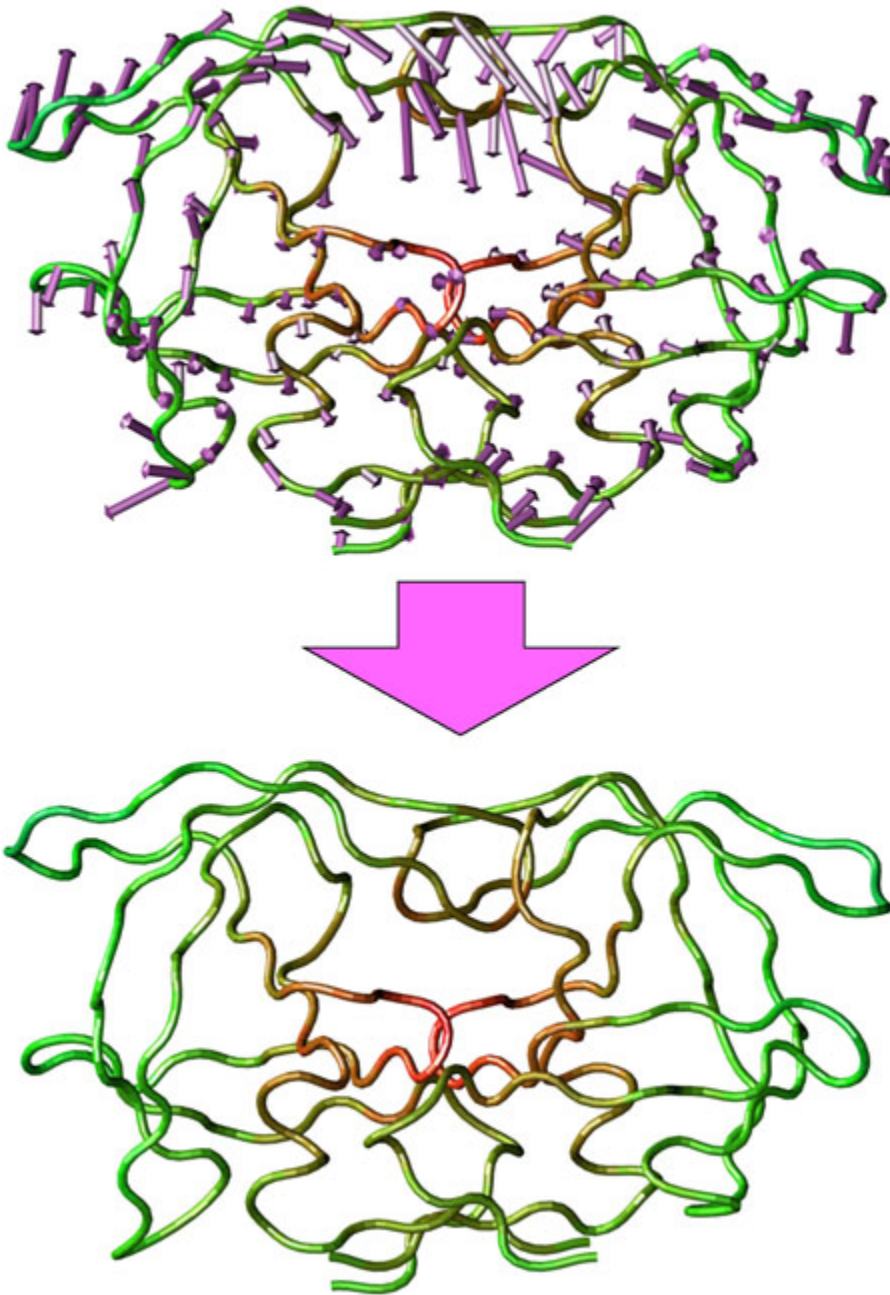


Molecular simulations can give a description of the full protein flexibility as it interacts with a ligand. Molecular dynamics applies the laws of classical mechanics to compute the motion of particles in a molecular system. Alternatively, the different conformational snapshots obtained at times  $t_0$ ,  $t_1$ , etc., can be used as multiple protein structures representing the conformational ensemble.

### Collective Degrees of Freedom

An alternative representation for protein flexibility is the use of collective degrees of freedom. This approach enables the representation of full protein flexibility, including loops and domains, without a dramatic increase in computational cost. Collective degrees of freedom are not native degrees of freedom of molecules. Instead they consist of global protein motions that result from a simultaneous change of all or part of the native degrees of freedom of the receptor.

Collective degrees of freedom can be determined using different methods. One method is the calculation of normal modes for the receptor [\[17\]](#). Normal modes are simple harmonic oscillations about a local energy minimum, which depends on the structure of the receptor and the energy function. For a purely harmonic energy function, any motion can be exactly expressed as a superposition of normal modes. In proteins, the lowest frequency modes correspond to delocalized motions, in which a large number of atoms oscillate with considerable amplitude. The highest frequency motions are more localized such as the stretching of bonds. By assuming that the protein is at an energy minimum, we can represent its flexibility by using the low frequency normal modes as degrees of freedom for the system. Zacharias and Sklenar [\[18\]](#) applied a method similar to normal mode analysis to derive a series of harmonic modes that were used to account for receptor flexibility in the binding of a small ligand to DNA. This in practice reduced the number of degrees of freedom of the DNA molecule from 822 ( $3 \times 276 \text{ atoms} - 6$ ) to approximately 5 to 40.



Representation of a collective degree of freedom for HIV-1 protease. Full protein flexibility can be represented in a low dimensional space using collective degrees of freedom. One method to obtain these is Principal Component Analysis. Principal components correspond to a concerted motion of the protein. The first principal component for HIV-1

protease is indicated by the arrows (top). By following this collective degree of freedom it is possible to generate alternative conformations for the receptor (bottom).

An alternative method of calculating collective degrees of freedom for macromolecules is the use of dimensional reduction methods. The most commonly used dimensional reduction method for the study of protein motions is principal component analysis (PCA). This method was first applied by Garcia [\[19\]](#) in order to identify high-amplitude modes of fluctuations in macromolecular dynamics simulations. It has also been used to identify and study protein conformational substates, as a possible method to extend the timescale of molecular dynamics simulations and as a method to perform conformational sampling. In the next module, we present a protocol [\[20\]](#) based on PCA to derive a reduced basis representation of protein flexibility that can be used to decrease the complexity of modeling protein/ligand interactions. The most significant principal components have a direct physical interpretation. They correspond to a concerted motion of the protein where all the atoms move in specific spatial directions and with fixed ratios in overall displacement. An example is provided in Figure 10 where the directions and ratios are indicated by the direction and size of the arrows, respectively. By considering only the most significant principal components as the valuable degrees of freedom of the system, it is possible to cut down an initial search space of thousands of degrees of freedom to less than fifty. This is achievable because the fifty most significant principal components usually account for 80-90% of the overall conformational variance of the system. The PCA approach avoids some of the limitations of normal modes such as deficient solvent modeling and existence of multiple energy minima during a large motion. The last limitation contradicts the initial assumption of a single well energy potential.

### **Recommended Reading and Resources:**

- A review of flexible receptor methods, as of 2003: Teodoro, M.L. and L.E. Kavraki. [\[HTML\]](#). (2003). Conformational Flexibility Models for

the Receptor in Structure Based Drug Design. Current Pharmaceutical Design, 9, 1635-1648.

- A comprehensive overview of existing docking software, as of 2006: Sousa, S.F., Pedro Fernandes A., and Ramos, M. J. [[HTML](#)]. (2006). Protein-ligand docking: Current status and future challenges. Proteins: Structure, Function, and Bioinformatics, 65(1) 15-26.

## Assignment 1: Visualization and Ranking of Protein Conformations

This assignment introduces students to the visualization software VMD and allows them to get familiar with protein structures. Students are required to visualize protein conformations and understand the differences between structures based on their geometry. Students will rank the provided conformations based on their geometric differences.

Please download files for this assignment from [here](#).

### Protein Data Bank

Your first task is to get familiar with the Protein Data Bank (PDB). Visit the [PDB website](#) and perform a site search for CI2 (chymotrypsin inhibitor2). Upon this search, some of the results (in particular those related to the Molybdenum Cofactor) you can disregard. Focus primarily on structures whose PDB name starts with "1C". Find the functional role of CI2 and answer [Q1](#). Your search will report many different experimentally resolved structures for this protein. Please answer why that is so in [Q2](#). Focus first on the resolved structures with PDB codes 1CIQ, 1CIR, and 1COA. Note that these structures were resolved from different organisms and with different experimental methods. Please answer [Q3](#).

Let's focus now on one particular structure, the one with PDB code 1COA. You can view an image of the structure of 1COA from the PDB website with the "Images and Visualization" frame to the right. You can either cycle through different images of the protein or download files for different programs and browser plug-ins. You can download the coordinates of the native conformation of 1COA at the "Download Files" option in the left frame. Download the structure file in PDB format. Please get familiar with the [PDB file format](#). For the purpose of this assignment, only the lines starting with the "ATOM" header are meaningful to us. Note that the x, y, z cartesian coordinates of each atom are stored in columns 7, 8, and 9. These coordinates specify the location of each atom of 1COA in 3D space. You will use such information in the future to visualize protein conformations.

### Visualizing Protein Conformations

For this section, you will need access to the VMD molecular visualization package. At the end of this assignment there are short instructions on installing VMD.

If you want to install VMD on a computer yourself, you can download a copy from its developers' [website](#). We recommend using the latest version. Installation varies with your platform. Windows versions are distributed as self-installing archives, and should work "out of the box." The MacOS-X version is distributed as a disk image that you simply unpack and move to the location of your choice. Linux and Unix versions will require you to compile the source code, but in most cases, this consists of switching into the correct directory and typing a couple of commands. For specific instructions on how to install the version for your machine, see the README file distributed with the installation archive, which includes a "Quick Installation Instructions" section.

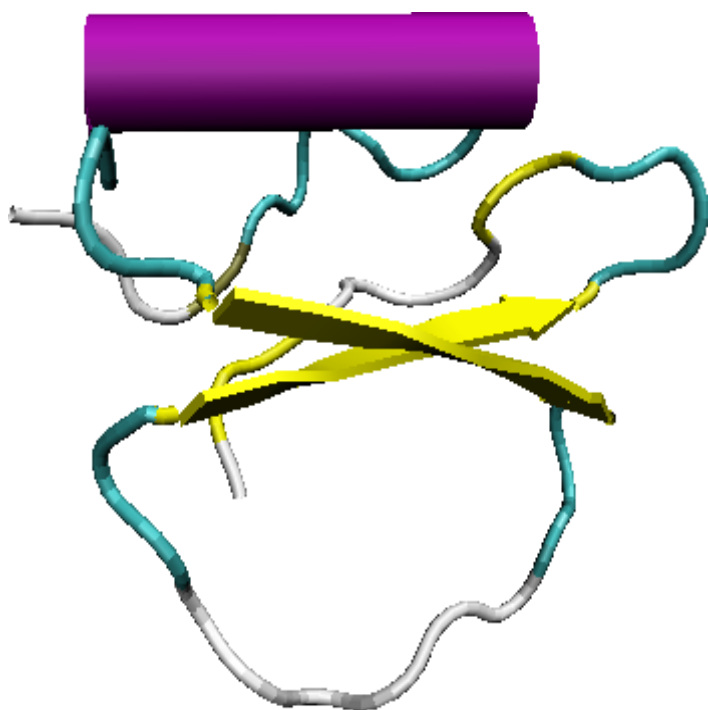
Your next task is to visualize the conformation specified in the PDB file for 1COA through VMD. VMD is a flexible visualizer that allows you to display, manipulation, animate, and analyze biomolecules in full-color, three-dimensional renditions. It also provides a built-in [Tcl/Tk](#) interpreter, allowing you to customize its behavior and add functionality.

Familiarize yourself with the capabilities of VMD through its [documentation](#), especially the User's Guide and various tutorials.

Start by visualizing the native conformation of the protein in 1COA.pdb. You can do so by selecting the "File/New Molecule" option in the "VMD Main" window. This will pop up the "Molecule File Browser" window. Browse through your files to find the file you downloaded from the PDB website and click "Open." Make sure the "Determine File Type" pulldown menu is set to "PDB," then click "Load."

You will note that the default graphical rendering represents bonds between atoms as lines. You can change the graphical representation of the molecule loaded through the "Graphics->Representation" menu option in the "VMD Main" window. Note that, by default, VMD selects all atoms for drawing. Please get familiar with the "Draw Style" options by experimenting with the "Coloring Method" and "Drawing Method" pulldown menus. One useful

drawing method is the cartoon representation, which draws  $\alpha$ -helices as cylinders and  $\beta$ -sheets as flattened arrows. You can further emphasize these secondary structure elements by choosing the "Structure" option under "Coloring Methods." You should be able to see something like Figure 1 below. You can save your rendering as an image through the "File->Render" option.



Native conformation of 1COA,  
"Cartoon" drawing, "Structure"  
coloring.

### A. Visualizing a Set of Conformations

Now it's time to look at more than one conformation. VMD supports reading of cartesian coordinates in multiple formats. We will use [crd](#) files from now on. A crd file contains one dummy line followed by a list of

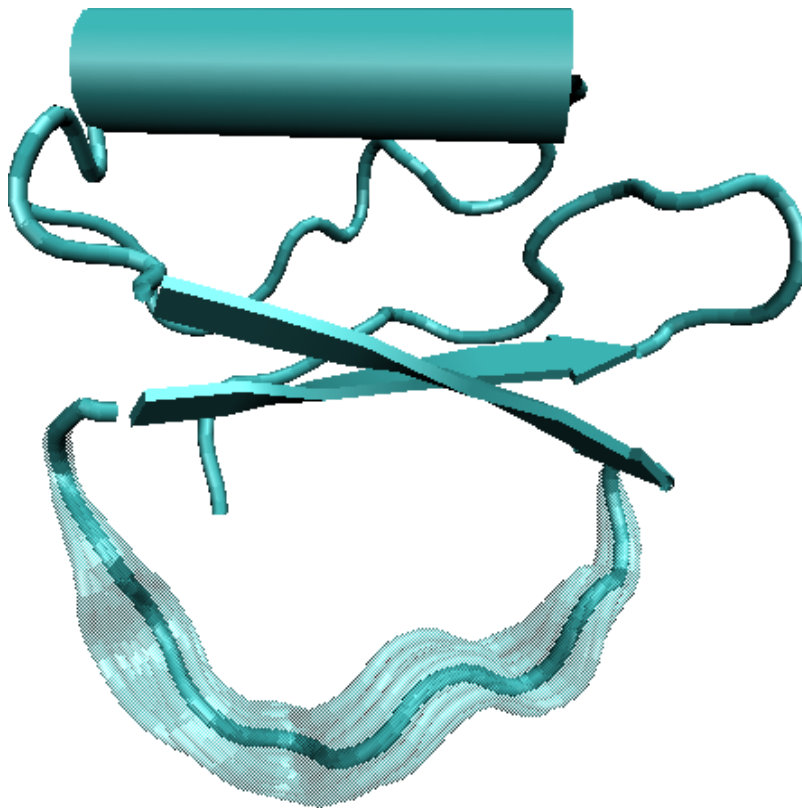
conformations, where each conformation is specified as a list of 3N coordinates for a protein of N atoms. There are other formats in which to store conformations, e.g. [dcd](#) files. Though they occupy less space because they store conformations in binary format, it is easier for you at this point to use crd files which store conformations in ASCII. VMD allows you to do i/o, i.e. you can read a crd file through VMD or write out the coordinates of a conformation to a crd file. Try VMD's output capabilities by writing out the native conformation specified in the pdb file for 1COA that you have already loaded at this point. Click with the mouse at the line that specifies the molecule you have loaded - this action will highlight it in yellow. Then follow the "File->Save Coordinates" option. At the new popped-up window, select the molecule for which you want to write out the coordinates (this should be 1COA), choose the crd file format, choose 0:0 for the beginning and end of the frames to be written out, and save the coordinates to a new crd file, e.g. "1COA\_native.crd."

In this assignment, you will be required to analyze conformations of 1COA. Since the file provided by the Protein Databank does not contain all atoms (hydrogen atoms are usually not reported in crystallographically-determined structures), we will be using a more complete version of the native conformation with the hydrogen atoms reconstructed: 1COA\_native.pdb, included in the .zip archive linked at the top of the page. Make sure to use this .pdb file from now on.

Clean up your VMD window by deleting or hiding the loaded molecule. To delete it, right click the molecule name in the "VMD Main" window and select "Delete Molecule." Now load the all-atom .pdb we provided. Right click the molecule name in the VMD window and select "Delete Frames." Click "Delete" in the window that pops up. Load the prepared .crd file 1coa\_confs.crd, which is included in the .zip archive linked at the top of this webpage, by selecting "File->Load Data into Molecule." The native conformation for 1COA is the first conformation specified in "1coa\_confs.crd" As the .crd file is loading, you will see VMD render each conformation in series. You can visualize this series of conformations as an animation, which is the default behavior of VMD, but in this case, it makes more sense to view the conformations superimposed over each other. You can superimpose conformations from the .crd file by using the "Graphical

Representations" menu's "Trajectory" tab, where you can tell VMD to draw multiple frames in the format a:b, where a is the number of the first frame you want to superimpose, and b is the number of the last frame (numbering starts at 0). You should be able to see something similar to Figure 2.

**Warning: If the machine you are working on is relatively old or slow, do not attempt to display more than 50-100 structures simultaneously. More than that could cause VMD to become unresponsive.**



1COA conformations superimposed

## B. Molecules in Motion

Molecular Dynamics simulations model the motion of a molecular system using a force field based on the chemical properties of the atoms involved, and updates its state at each moment in time using Newton's Laws of Motion (classical mechanics). You will now visualize the conformations visited by HIV-1 protease during a short MD simulation. The corresponding native structure is available in `hiv_native.pdb`, in the .zip archive for this assignment. Load the native structure .pdb and then the trajectory (.crd) file `hiv_md_simulation.crd` in [missing\_resource: assignment.zip]

Note that you can always save your work session in a state .vmd file. This can be very convenient if you do not want to repeat all the graphical representations you have already chosen for 1COA. Please visit the manual for details on how to save and resume a session.

## Ranking Conformations

The conformations you superimposed on one another look different from the native conformation. We will look at one possible way to quantify the difference or similarity between two conformations: LRMSD (Least Root Mean Squared Deviation). This is a measure of the average of the distance each atom would have to move to convert one structure to the other. In this assignment you will use a standard script that computes the LRMSD between the conformations already loaded in a trajectory in VMD from a reference conformation. Please download the [Tcl script](#) to compute the RMSD of conformations from a reference conformation. You may want to modify the script to print to a file of your choice. You can do this by modifying the line that opens the file (`set fp [open "rmsd_output.txt" w]`). If you prefer to print to the screen, you can increase the buffer size of the Tk console in VMD typing `tkcon buffer xxxx` inside the console itself, where `xxx` is the number of lines to store. As an alternative, you may use the Matlab script `rmsd.m` included with the files for the assignment. Note that this script requires the other three .m files to be in the same directory, and that it assumes that the first line of all .crd files is a comment that can be discarded. Please answer [Q4](#).

## Visualizing Protein Substructures

In this section, you will investigate the evolutionary conservation of protein substructures such as enzymatic catalytic sites. You will use publicly available online databases in addition to VMD to analyze the structures of two peroxidase enzymes and understand different methods of classifying related protein structures.

## **Structurally Classifying Proteins**

There are many classification systems for grouping similar protein structures within the PDB. The Enzyme Commission (EC) nomenclature classifies enzyme structures by their catalytic function into 4 increasingly specific levels. The heme-dependent peroxidases that you will investigate are a family of enzymes found in plants, fungi, and animals and the 4-part hierarchical EC number of the family is 1.11.1.7.

### **Understanding EC classification**

Start by visiting the PDB and searching for "1.11.1.7". Your search will reveal many structure entries in the PDB that belong to the EC family of heme-dependent peroxidases, such as myeloperoxidases and lactoperoxidases found in higher animals. Click on the "EC" icon (a small, white circular icon near the EC number for each entry) for one of the entries in the search results; this will take you to a description of EC 1.11.1.7. The EC number page provides information on principal references for the family, alternative names for enzymes in the family, and links to additional enzymatic databases such as BRENDA and KEGG. At the bottom of the page, you will see links to higher-level EC classifications such as "EC 1.11.1", "EC 1.11", and "EC 1". Visit each of the pages to understand the more general functional classes to which EC 1.11.1.7 belongs. For example, EC 1.11.1 contains many different families of peroxidases including and in addition to EC 1.11.1.7. [Q5](#): name 2 other peroxidase families from EC "1.11.1".

Visit the page for EC 1 (link at the bottom of the EC 1.11.1.7 page). EC 1 is the most general level of functional classification for the heme-dependent peroxidases. There are 6 major EC classifications including EC 1 (EC

1,2,3,4,5, and 6) and the heme-dependent peroxidases belong to the top-level classification of Oxidoreductases.

### **Understanding CATH and SCOP classification**

Return back to the PDB and search for entry 1CXP which will be a human myeloperoxidase belonging to EC 1.11.1.7. The PDB entry page for 1CXP lists additional structural classification data about the enzyme at the bottom of the page. Scroll down to the section titled "SCOP Classification". The Structural Classification Of Proteins (SCOP) is another system for categorizing protein structures that, like the EC system, uses a four level hierarchic system. The four SCOP levels correspond to structural class, fold, superfamily, and family. The superfamily of 1CXP as defined by SCOP is listed as "myeloperoxidase-like". Below the SCOP for 1CXP is the CATH structural classification information for the enzyme. CATH stands for Class (C) Architecture (A) Topology (T) Homologous superfamily (H) and provides an additional classification scheme for the structure. The class and architecture classification for 1CXP as defined by CATH are "mainly alpha" and "orthogonal bundle" respectively. The CATH class describes the major secondary structure composition of the enzyme, and therefore "mainly alpha" indicates that 1CXP is composed primarily of alpha helices.

### **Comparing structural classifications of fungal (1ARU) and human (1CXP) peroxidases**

Now you will compare the structural classification of the human 1CXP to a fungal heme-dependent peroxidase in order to learn how to analyze the structural similarities and differences between the two similar functioning enzymes from two very different organisms. Search the PDB for "1ARU", a peroxidase structure from *Arthromyces ramosus*, a fungal species. [Q6](#) Which levels of the SCOP and CATH classifications do 1ARU and 1CXP have in common, and at what level of classification do they begin to differ? You will see that while the two enzymes are functionally related, they have significant structural differences. As a side note, the sequence similarity of the two enzymes is less than 10%, and 1CXP is composed of two chains, while 1ARU is composed of only a single amino acid chain.

Now you will visualize the two peroxidase enzymes to examine a common substructure shared between them within the catalytic site. Download the

PDB structure files for both 1ARU and 1CXP from the PDB.

### **Draw the protein backbone in VMD**

First open 1ARU.pdb in VMD. There are many different ways of visualizing the protein. Open the molecular representation menu of VMD by clicking "Graphics" and then selecting "Representations..." from the menu, which will open a new menu window. Change the "Drawing Method" to "NewCartoon" to draw only the backbone of the protein and display all alpha helices in helix form. Change the "Coloring Method" to "ResType", which will color each amino acid in the backbone by the residue type to which it corresponds: non-polar residues (white), basic residues (blue), acidic residues (red) and polar residues (green).

### **Draw the catalytic substructure**

Now you will highlight a substructure within the catalytic site of the 1ARU. In the "Graphical Representations" window where you previously changed the drawing type of the whole protein, click the "Create Rep" button to create a new representation. Select the new representation (the selected representation will be highlighted in yellow within the menu). Now change the text in the "Selected Atoms" field from its current value ("all") to "resid 52 56 93", which will select only residue numbers 52, 56, and 93 (you will not see any changes in the visualization window yet). Now so that you can distinguish these amino acids from the rest of the protein, change the "Drawing Method" to "DynamicBonds" and the "Coloring Method" to "Name". You will now see the side chains of these catalytic site amino acids within a pocket/cleft of the protein.

### **Draw the Heme group**

There is also a heme group within this pocket near the catalytic amino acids that we will highlight now. Create another molecular representation by clicking "Create Rep" again. Select the new representation. Change the "Selected Atoms" to "resname HEM" which will select the only heme group within the protein. You will see that the heme group is co-located with the amino acids you have just highlighted. Rotate the molecule and zoom in on the catalytic site that we have just revealed. Output a picture of the catalytic site for your submission by going to "File -> Render..." and

then selecting "PostScript" from the "Render using" menu. One free utility for viewing PostScript files is GhostView.

### **Compare fungal (1ARU) and human (1CXP) proteins**

Now you will repeat the process for 1CXP, except that the amino acids for the catalytic site should be selected with "resid 91 95 239". Again output a picture of the catalytic site, this time for 1CXP. [Q7](#) Examine the structures of 1CXP and 1ARU in VMD and list 3 differences between the enzymes that you see.

## **For Submission**

Please follow this list closely.

### **Deliverables**

- **Q1** Describe in one paragraph what the role of CI2 is and in what reactions it is involved. You may use any source (including the PDB).
- **Q2** List some reasons for having more than 16 different structures for CI2.
- **Q3** List the experimental methods used for resolving 1CIQ, 1CIR, and 1COA, the organisms from which these structures were extracted, and the respective resolutions reported (some of this information may be inside the PDB file, but some may be found in the structure's page within the PDB). Please explain why reporting the resolution is essential to resolving the structure for a protein.
- **Q4** Compute the LRMSD of all the conformations provided in 1coa\_confs.crd with respect to the very first conformation in the same file. Plot their LRMSD on the Y-axis and the conformation ID on the X-axis. You can use Excel, Gnuplot, SM, Matlab, or any other plotting software. Please submit this plot.
- **Q5** Name two protein families, as defined by EC nomenclature, within the "1.11.1" classification.
- **Q6** In terms of CATH and SCOP classifications, how are 1ARU and 1CXP similar and where do they differ?
- **Q7** With VMD (or the visualization tool of your choice), create an image for each of the two corresponding catalytic substructures, one

within 1ARU and one within 1CXP. List 3 differences that you see between the structures 1ARU and 1CXP.

Note: While you are welcome to use any method (MS Word, LaTeX, etc.), **please typeset your deliverables**. Although we have assumed you are using VMD in writing the assignment, you are free to use any visualization software for molecular structures or even write your own. Only VMD will be supported by the TA, however.

## Appendix: Installing VMD

VMD can be installed easily in Windows/Linux/Solaris by getting the latest version from [here](#). Just click on the "Download VMD" link on the left and choose the right platform. The page will ask you to register if it's the first time, so simply choose a username and password and accept the usual "no-commercial-use" license agreement.

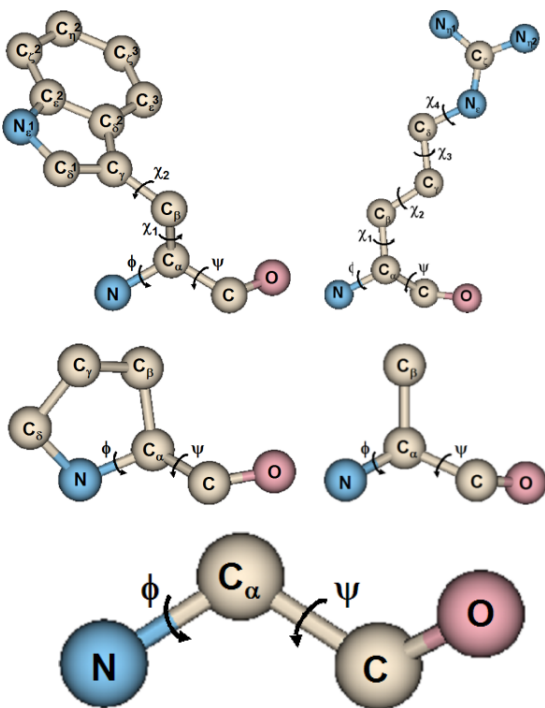
If you want to use VMD in the OwlNet Windows machines, you will need to install it locally, so get the Windows VMD installer from the link above and execute it. It has an intuitive graphical installation process.

If you use Linux/Solaris (either at home or an OwlNet machine) you can get the appropriate version and install it. The UNIX distributions usually are tarball files. After you download them, you can un-tar them with: `tar zxvf vmd-1.8.5.bin.PLATFORM.opengl.tar.gz` Which will create an installation directory. CD to this directory and follow the short instructions in the README file.

## Assignment 2: Performing Rotations

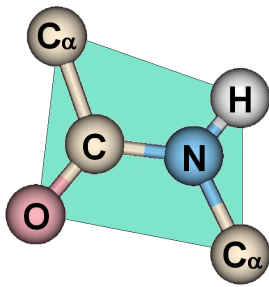
### "Defining the Connectivity of a Backbone Chain"

Figure 1 provides examples of some aminoacids, from long ones such as arginine to bulky ones such as proline, and to the smallest one, glycine. Note that as we have mentioned, while different in their sidechain atoms, all these aminoacids share a common group of atoms that includes N, CA, C, and O. The chain that goes through this shared core is known as the backbone. For the purpose of this module, you will consider as the backbone the chain that goes through N, CA, and C for every aminoacid. Note that there are two backbone dihedrals per aminoacid, with the number of sidechain dihedrals varying depending on the aminoacid. To simplify the rotation by dihedrals, in this assignment you will only manipulate protein structures from which the sidechain atoms have been removed. This means that you will only deal with a backbone chain that goes through the N, CA, and C of every aminoacid.



Sample Aminoacids

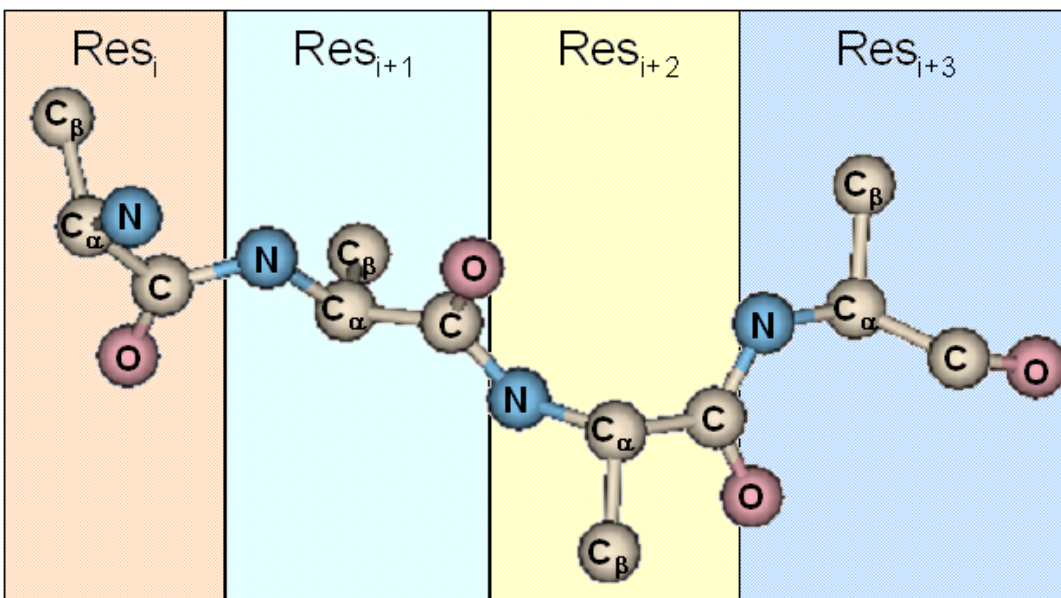
You should be familiar with how aminoacids connect to each other through the peptide bond. Figure 2 illustrates the peptide bond formed between the C of an aminoacid and the N of the following aminoacid. Consecutive applications of the peptide bond create the backbone chain. Recall that due to its double covalent nature, the peptide bond is planar (illustrated with the plane in Figure 2), and unrotatable. Therefore, in your manipulation of the backbone chain, you do not need to consider the peptide bond for rotation. You are left with two dihedral bonds per aminoacid.



Due to its  
double  
covalent  
nature, the  
peptide bond  
is rigid.

Figure 3 shows four consecutive aminoacids in a polypeptide chain. Note the peptide bonds that connect the aminoacids. You need to understand that rotation of a backbone dihedral will affect the location of all atoms following it due to the accumulation of rotations along the backbone. Therefore, it is necessary for you to define an orientation for the backbone. The easiest orientation is that used for the sequence of a protein, where the backbone is defined as the chain starting at the N atom of the first

aminoacid of the sequence and ending at the C atom of the last aminoacid of the sequence. For your convenience, the native conformation supplied to you later in this assignment will list the coordinates of the atoms in the order N, CA, C for every aminoacid. So, reading in order from this file will give you the orientation of the backbone.



A series of applications of the peptide bond give rise to the polypeptide chain.

Before you proceed to the next section, make sure that you can answer these questions:

- If we rotate the dihedral bond between the N and CA of the last amino acid, which atoms' locations will change in 3D space? Answering this question will help you answer [Q1](#).
- Why do we usually say that rotations by dihedrals produce large scale motions? Think of an example of a rotation by dihedral where this is true. Answering this question will help you answer [Q3](#).

## Dihedral Rotations

Now it is time to actually rotate dihedral bonds. Recall from our discussion of [Forward Kinematics](#) that one can define the transformation matrix/matrices to account for a dihedral rotation, depending on the representation method you choose (global or local coordinates, as discussed below). If you need to compute values such as bond lengths and bond angles, You can compute the bond length between two atoms as the Euclidean distance between them. You can also compute the angle between two bonds, the bond angle, as the the angle between two [normalized](#) vectors, which amounts to taking the [dot product](#) of the normalized vectors. If you need to identify one atom as your anchor atom, with the default backbone orientation, your anchor atom is the N of the very first aminoacid (the very first atom of the chain).

Now we will put our knowledge into practice. You will manipulate the structure in the pdb file [missing\_resource: backbone\_native.pdb]

## Setup with Matlab

Even though you are welcome to use any programming language to perform rotations, here we provide you with a setup for Matlab. Matlab offers a lot of matrix operations that you would otherwise have to implement yourself in languages like C/C++, for example. We provide some hints/directions for those of you who choose to implement this assignment in Matlab. The very first step you need to do is to read from an ASCII file that contains the cartesian coordinates of the chain you will manipulate and save these coordinates in a data structure that will represent your chain. The simplest data structure at this point is an array, where positions  $3*(i-1)+1$ ,  $3*(i-1)+2$ , and  $3*(i-1)+3$  contain the coordinates of atom  $i$ . Note that Matlab starts counting from 1. Take a look at the backbone\_native.pdb file. Find out what the number of atoms is. You can read the cartesian coordinates from the [backbone\\_native.crd](#) file (note that this .crd file does NOT have a dummy line at the beginning to read it easily with Matlab). You can do so with the command:

```
cartesian_coordinates = textread(input_file,
```

`'%f');` where you have set the input file to where you have stored `backbone_native.crd` as in for example `input_file = '/home/user_name/rest_of_the_path/backbone_native.crd'`; . You can check how many coordinates you have read with the command `length(cartesian_coordinates)` .

The `textread` command stores all coordinates read from the `backbone.crd` file in the `cartesian_coordinates` array. You could manipulate this array with the dihedral rotations you will define. However, it might be more convenient for matrix operations and for clarity to store the cartesian coordinates in a matrix with N rows and 3 columns for each row. In this way, row i contains all the three cartesian coordinates for atom i. Define the number of atoms in the backbone with the command `N is length(cartesian_coordinates)/3;` . Now you need to declare a matrix with the right dimensions. Since you actually need to work with homogeneous coordinates, it might be convenient to declare a matrix with N rows and 4 columns, where the last column contains 1. You can do so with the command `backbone_chain = zeros (N, 4);` which creates a matrix with N rows and 4 columns initialized to all zeros. You can set the fourth column to 1 with the command `backbone_chain(:, 4) = 1;` Now you need to place the cartesian coordinates from the array to this matrix. You can do so with the for loop below: `for i = 1:N for j = 1:3 backbone_chain(i,j) = cartesian_coordinates((i-1) * 3 + j); end; end;`

The cartesian coordinates you have just read will serve as a basis for performing rotations, according to what method you choose to represent and manipulate the protein, as discussed in class. The following guidelines should help you get started with either method:

- If using the global coordinate (simple) approach, you can use the cartesian coordinates just read as a basis to perform rotations on them. For the most interesting task of transforming atomic positions, you simply need to iterate over the atoms affected by the transformation and multiply their position with the transformation matrix. Say the position of atom i needs to be transformed by your transformation matrix. You can do so by multiplying `trans_mat` with

`backbone_chain(i, :)` as in `trans_mat * backbone_chain(i, :)`'. Note that the colon is very convenient as it gives an entire row or an entire column and that `backbone_chain(i, :)`' gives you the transpose that is necessary for the multiplication to be carried out.

- If using the Denavit-Hartenberg local frames, you first need to extract the bond lengths and angles, and the initial dihedral angles, from the cartesian coordinates. To do so, you can loop over the atoms computing these values and storing them (for example in arrays named `bonds, angles, dihedrals`). Once this is done, you have extracted a representation of the protein in its internal coordinates and can discard the cartesian coordinates, or keep the coordinates array to store the reconstructed protein after performing manipulations. The absolute position/orientation of the protein is not important and can be ignored by assuming the anchor atom rests at the origin and that the first bond angle and dihedral angle are both zero. Remember you can perform rotations now by simply adding/subtracting from the dihedral angles, but to recover the cartesian coordinates once the rotations are done you need to build a chain of homogeneous transformations as discussed in class.

In any case, your transformation matrices need to have dimension 4x4 to work with homogeneous coordinates. You can initialize an empty 4x4 matrix by writing `trans_mat = zeros(4,4);`. Then you can set the elements of this matrix to what they should be for the particular method you use. You can evaluate all the cosines and sines that you need in Matlab. Matlab offers you built-in operations such as dot product or vector norm. Make sure that you understand these operations before you apply them. Recall that you can rotate any bond except the peptide bond, which is rigid.

### Conduct the following rotations:

- Rotate the dihedral bond between the N and CA of the last aminoacid by 30 degrees. Does the structure change much? What atoms move in space as the result of this rotation? Please answer [Q1](#).
- Now choose any dihedral bond roughly in the middle of the protein and experiment with its rotation until you reach a conformation that

has steric clashes (atoms collide with one another). Please answer [Q3](#) and [Q4](#).

If you wish to visualize the resulting conformations in VMD, you simply have to produce a file with the transformed coordinates. This file should be in ASCII (text) format and must have an empty/dummy first line, and all atom coordinates following (for example all in a single line, separated by spaces). You can save this file with ".crd" extension. Then, you can use the same "backbone\_native.pdb" file, and use VMD's option "Load data into molecule" to load your new coordinates from the .crd file.

## Ranking by Energy

Protein conformations are not only geometric objects but are characterized by energetic stability. You have seen that there are many empirical ways to compute the potential energy of a conformation. Functions such as [CHARMM](#) are very involved for the purpose of this assignment. Therefore, let us not worry about all the energetic terms to be considered from the atomic interactions. Consider only unfavorable interactions due to collisions between atoms, also referred to as steric clashes. Think about a simple energy function that checks for steric clashes. Your function should report high energies for conformations with collisions and low energies for collision-free conformations. You can model each atom as a sphere with a certain radius known as the Van der Waals (VDW) radius. Even though different atoms have different VDW radii, you can assume that all atoms have the same radius of 1.7 Å. To assist you in devising a function that checks for collisions over all pairs of atoms, consider the following:

$$\sum_{i,j} (C_i - C_j)^2 > (2 \cdot R)^2$$

For all different  
atoms i and j, make  
sure that the  
distance between

their centers C is  
more than twice  
their radiuses. In  
this simple example  
we are using the  
same radius R.

## Evaluating Rotations by Energy

- Now think about the effect of the dihedral rotation of the last bond the energetic stability of the new conformation. Please answer [Q2](#).
- When you rotated a dihedral bond in the middle of the backbone until you reached a conformation that had collisions, you verified the presence of steric clashes by visualizing the conformation. You can quantify the presence of steric clashes now through your energy function. Please answer [Q5](#).

Again, as in dihedral rotations, you are encouraged to use Matlab. After you have the cartesian coordinates of the manipulated conformation, you can iterate over the atoms and check whether they are in collision with one another. You can do this with a double loop. Assume that the radius of every atom is the same, 1.7 Å. In order to avoid reporting collisions for bonded atoms, you can consider only pairs of atoms that are 4 positions apart. That is, check atom  $i$  with atom  $i+4$  for a possible steric clash.

## For Submission

NOTE: remember that in order to open a .crd file in VMD (to attach it to a loaded structure) it has to contain a dummy first line. If your implementation writes out ASCII coordinates, remember to add an empty/dummy line at the beginning before opening with VMD. Please follow this list of deliverables closely:

### Deliverables

- **Q1** Superimpose the conformation resulting from rotating the dihedral bond between the N and CA of the last aminoacid by 30 degrees over

- the native backbone conformation. Prepare an image that clearly shows which atoms' locations have changed. Submit this image.
- **Q2** Now you need to quantify the energetic cost of rotating this dihedral bond by 30 degrees by reporting whether this rotation causes any steric clashes. Please test your energy function on the conformation you obtained from rotating the last dihedral bond by 30 degrees. Please report the answer of your energy function. It should be compatible with what you can visualize from the image you prepared above.
  - **Q3** Provide the index of the dihedral bond you need to rotate to generate a large scale motion that results on collisions between atoms. Report on the amount of rotation that you need to perform in order to obtain a conformation that contains steric clashes. Plot this conformation superimposed on the native backbone conformation so as to show that it is very different from the native. Render this image and submit it.
  - **Q4** Zoom in on that part of your conformation that contains collisions. Make sure that this image clearly shows steric clashes and submit this image.
  - **Q5** Compute the energy of the above conformation with your energy function. Report the answer that your energy function returns. Make sure that this answer reflects the fact that there are steric clashes. Please provide the atomic distances for the atom pairs that your function reports in collision.
  - **Bonus** Upon a situation when the energy of the conformation resulting from a dihedral rotation is high, how would you minimize the energy of this conformation? Provide a discussion and pseudo-code on how you would perform the minimization.

Note: While you are welcome to use any programming language, please typeset your deliverables and make sure that the quality of your images is good. You can use any visualization software to produce your images. You are welcome to perform the asked rotations either through the Denavit-Hartenberg derivation provided in our [\[PDF\] document](#), or through simple rotations around arbitrary vectors. If you choose the latter, please note that there is a typo in the formula for the rotation matrix  $R_i$  in [Zhang-Kavraki, 2002 \[PDF\]](#), on page 2: The second column, first row entry of this matrix is

given as  $u_x v_y 1 - \cos(\theta) + v_z \sin(\theta)$  , but it should be  $v_x v_y 1 - \cos(\theta) - v_z \sin(\theta)$  .

## Assignment 3: Inverse Kinematics

### Motivation for Inverse Kinematics in Proteins

One important application of inverse kinematics is in determining missing portions of protein structure. We traditionally depend on experimental techniques to provide us with the picture of an average structure for a protein. X-ray crystallography for example, relies on crystallizing proteins and reporting the structure of the protein crystal within a certain resolution. One of the inherent problems with X-ray crystallography is that mobile protein regions such as loops cause disorder in the crystal and as a consequence, coordinates for the atoms of these mobile regions cannot be reported. Often, in the PDB, crystallographically determined proteins are partially resolved, i.e. a portion of the structure may be missing due to its intrinsic mobility. Even when experimental techniques such as NMR and cryo-EM can report an average picture of the fully resolved protein, the average structure reported is not indicative of the different conformations mobile regions can assume inside our cells at room temperature.

The specific problem of completing a partially resolved protein structure by finding conformations for its missing loop is known as the fragment completion or the loop closure problem. Note that the loop closure problem is actually an inverse kinematics problem. Using sequence information alone, i.e. knowing the aminoacid sequence of the missing loop, one can generate starting loop conformations. The loop closure problem requires these loop conformations to be geometrically constrained by attaching them to the portion of the protein structure that is experimentally determined. Note that, as the picture below indicates, one can generate many loop conformations in space through forward kinematics. One end of the loop can be attached to its counterpart in the protein through translation alone. The other end however, needs to be attached without breaking bonds or stretching bond angles. One way to do this is through inverse kinematics; that is, knowing the goal position in space for the end of the loop, can we solve for the dihedral DOFs of the loop conformation? This question can be answered by Inverse Kinematics techniques.



One "sticky" end of the loop can be attached to its stationary counterpart in the protein through translation. The other end needs to move towards its goal location by solving an Inverse Kinematics problem.

## Inverse Kinematics for a Polypeptide Chain

### Cyclic Coordinate Descent (CCD)

In this assignment you will complete a loop portion in the 1COA structure of the CI2 protein. X-ray crystallography completely resolves the 1COA structure. However, even though a long loop region from residue 34 to residue 46 is present in the native conformation of 1COA, an interesting exercise is to pretend this loop region cannot be determined. Using an inverse kinematics approach, you will sample many potential loops that can all complete the 1COA structure and compare them to the native conformation obtained through X-ray crystallography. An important question to answer is whether your loop closure algorithm can recover/predict the conformational state of this loop region in CI2; that is, how different are your loops from the one in the native conformation of CI2? You will implement a simple inverse kinematics technique, Cyclic Coordinate Descent (CCD) as presented in [\[link\]](#). For simplicity, you will work with the native [missing\_resource: backbone\_native.pdb] [backbone\\_native.crd](#) file.

Your task is to generate 10 different loops that all complete the protein structure of CI2 and compare them to the native loop conformation. To do so, even though the native loop conformation is given to you in the

[missing\_resource: backbone\_native.pdb][[link](#)] to steer these loop conformations to their goal/target location as specified by the coordinates of residue 46 in the native conformation of CI2.

The Cyclic Coordinate Descent, as defined in [[link](#)], allows you to find the optimal dihedral angle for a dihedral bond by whose rotation a desired atom will get as close as possible to a target position. To simplify this problem, you will not have target positions for the three atoms N, CA, C as [[link](#)] does, but only on one atom, the C atom. This means that the equations you need to solve for the optimal dihedral rotation are going to be much simpler, with your distance measure  $S$  defined as the Euclidean distance between current position  $M$  of atom C and its target position  $F$ . Please address [Q1](#).

### CCD for a Chain

You have noticed by now that CCD reports the angle by which to rotate a particular dihedral bond so that the feature atom gets as close as possible to its target position. Consider a chain with the feature atom at its end. As in real polypeptides, this chain does not contain only one dihedral angle, but many. How would you use the CCD routine to solve for all the dihedrals of this chain so that the final end of the chain reaches a target position? Please address [Q2](#). (Hint: You might consider solving for each dihedral one at a time, after you have imposed an order on the dihedrals you are working with.)

For convenience, we will refer to the conformation of the chain where the atom reaches its target position as the **closure conformation**. The problem you will address in this assignment is how to obtain different closure conformations by solving for the dihedral DOFs of a chain in order to go from a starting/initial/current conformation to a closure conformation. In order to generate closure conformations, before solving for the dihedral DOFs through CCD for a chain, you need to generate starting/initial conformations for the loop region of CI2. One way to do so is to generate random angles for the dihedrals of the chain and then accumulate the respective transformation matrices down the chain through Forward Kinematics, as you have done in Assignment 2. You are not allowed to rotate only one dihedral bond to generate a **random initial transformation**.

You have to rotate all dihedral bonds of a chain by random angles in the interval  $[-\pi, \pi]$ . Please answer [Q3](#).

Now it is time for you to put your pseudocode to the test. We first list the particular specifications for the inverse kinematics problem you will solve in this assignment.

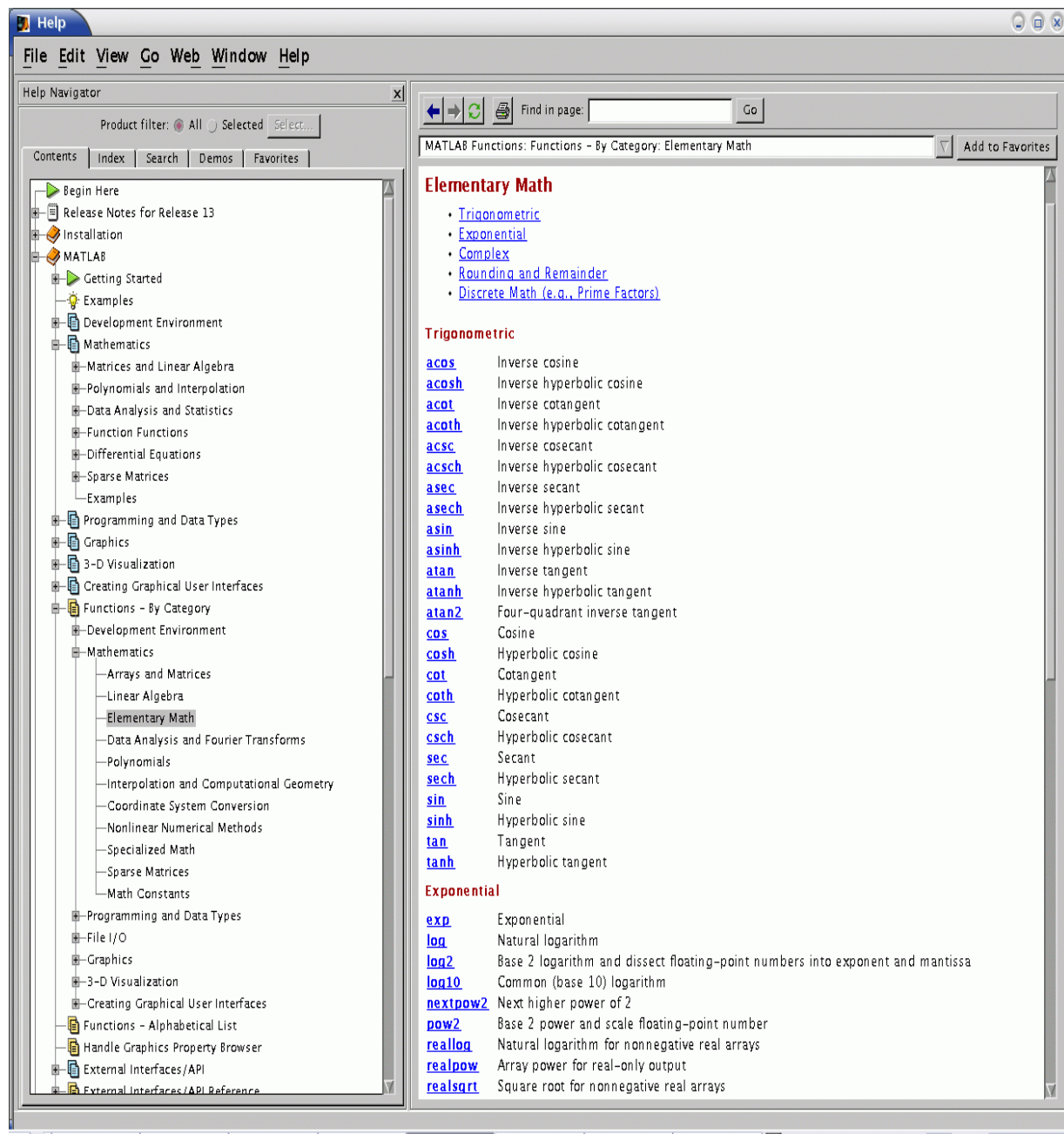
- For this problem, you will steer the C atom of residue 46 of the loop region towards its target position. We will refer to this atom as the **feature atom**.
- We need to define the DOFs. As you will only generate closure conformations for the loop region from residue 34 to residue 46, the only DOFs you can sample in the initial loop conformations and solve for in the closure conformations are those of the loop region. Every other dihedral you cannot change.
- The CCD algorithm needs to have a measure of distance between the current position of the feature atom and its target position. After you have generated initial conformations for the loop region, the current position of the feature atom is position in space of the C atom of residue 46 in this initial conformation.
- The target position of the feature atom will be its position as specified in the original (native) conformation contained in the [backbone\\_native.crd](#) file.
- Now we can put the pieces together: Generate an initial conformation by rotating the dihedrals of the 34-46 loop region of CI2 with random angles sampled uniformly in the  $[-\pi, \pi]$  interval. Solve for the dihedrals of the 34-46 region so that the C atom of aminoacid 46 in the initial conformation you have just generated reaches its target position as specified by its position in the native conformation in the backbone\_native.crd file. Note that depending on the random conformation you start with, it might be hard to steer atom C to reach its target position. After applying the dihedrals you have solved for, you might need to do additional iterations to solve for the dihedrals of the updated conformation, apply those and so on until the spatial constraint for atom C is satisfied. To have a quantitative criterion for when atom C satisfies its spatial constraint, you need to define a

threshold parameter for S, as in [\[link\]](#). Set this threshold criterion to 0.01 for this assignment. Please answer [Q4](#).

In this way, you will generate 10 closure conformations where the C atom of aminoacid 46 will be in its target position, its position in the native conformation. It is interesting to note that the native conformation itself is a closure conformation since it satisfies the spatial constraint on this atom. Now you can quantify the similarity/difference between these 10 closure conformations and the native conformation. As in assignment 1, you can compute the least RMSD of these 10 closure conformations from the native. You can also compute the energy of each of these 10 conformations and the native. Please address [Q5](#) and [Q6](#).

## Setup with Matlab

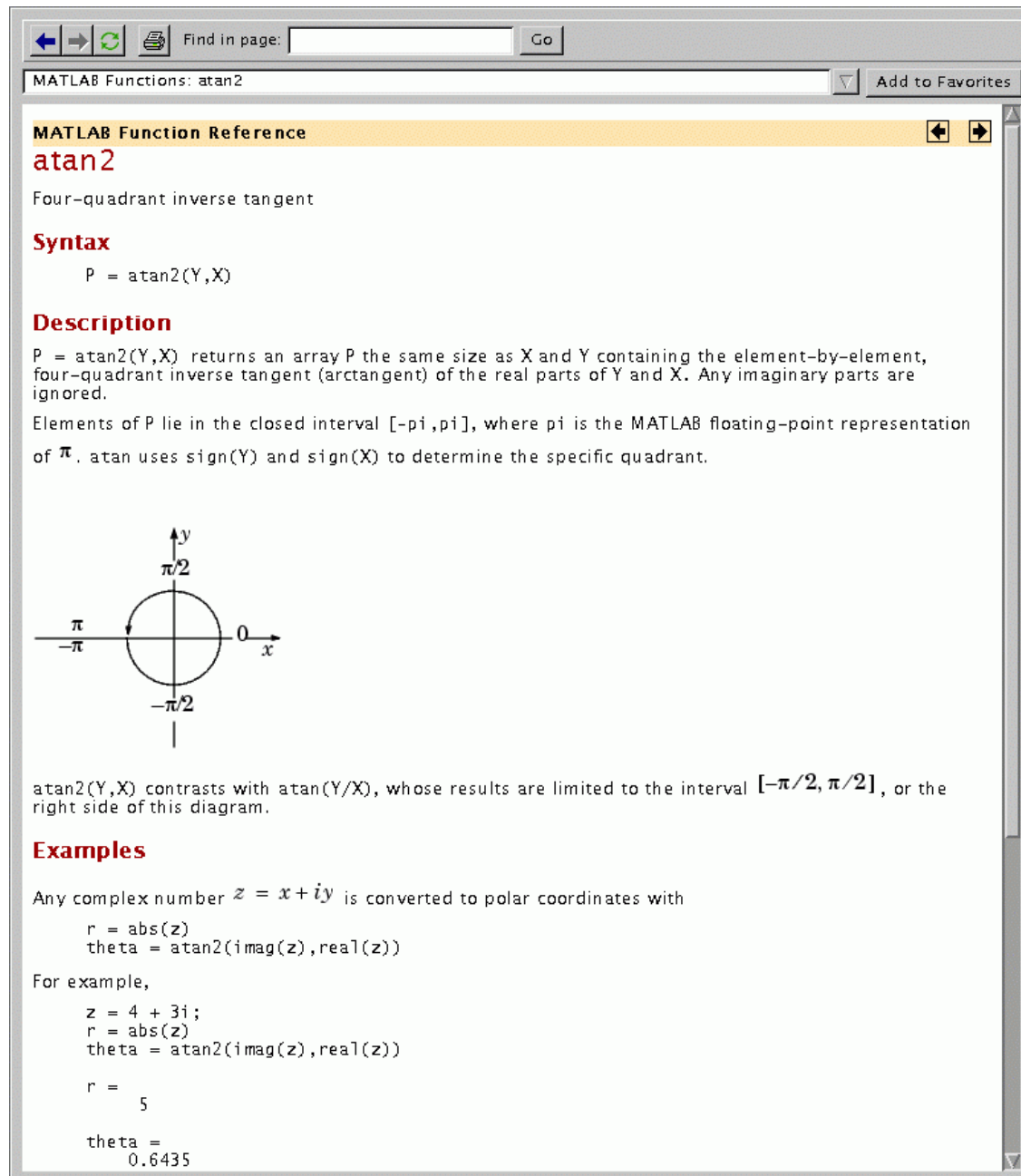
Even though you are welcome to use any programming language to complete this assignment, we suggest that you use Matlab due to its convenient matrix operations. For this assignment you will need some basic trigonometry functions such as [sine](#), [cosine](#), [tangent](#), and [inverse tangent/arctangent](#). You can find all these functions already implemented in Matlab. Please get familiar with their syntax by using the Help pages. Figure 1 captures the Help page for the category of **Functions By Category**, subcategory **Elementary Math**. You need to read this category to learn more on how to use these functions.



## Matlab Trigonometry

It is very important that you use the right arctangent implementation for this assignment. The authors in [\[link\]](#) use the `atan2` function as implemented in `C`. There is a similar function in Matlab. Please be careful to understand in what quadrant this function returns its output. You can learn

more about this function under the Help pages. Figure 2 captures the help page for this function under the category of **Functions By Category** , subcategory **Elementary Math** , as part of the **Trigonometric** functions.



Matlab `atan2` function

## For Submission

Please follow this list closely.

### Deliverables

- **Q1** Please repeat the derivation in [\[link\]](#) with the simplification we made that only one atom is the one we want to move to its target position. Typeset these derivations for submission.
- **Q2** Please typeset your pseudocode for how you would iterate over a chain until the end of the chain reaches its target position.
- **Q3** Please provide pseudocode for how you would generate a random conformation for the 34-46 region of the protein. Please generate 10 conformations in this way and visualize these conformations through VMD, all superimposed over the native. Submit this image.
- **Q4** Please plot all the final 10 conformations where the C atom of aminoacid 46 reaches its target position. Plot these conformations over the native. Indicate where the C atom lies with a VDW representation in red. Submit this image. (Note: remember that VMD requires a dummy first line in .crd files, so add one if your code does not).
- **Q5** Please compute the RMSD of the 10 closure conformations from the native. Please plot these values and submit this plot. Please check the 10 closure conformations and the native conformation for collisions with the energy function you implemented in assignment 2. Please report the energetic scores of each conformation. Does the native conformation contain any collisions? How do the other closure conformations compare to the native in terms of energy? What can you conclude about your implementation of CCD to get a closure conformation with regard to the energy of this closure conformation?
- **Q6** The CCD algorithm does not take into consideration energetic constraints due to unfavorable steric clashes. One way to impose energetic considerations is to discriminate against closure conformations that contain steric clashes. You have already implemented a simplistic energy function that can report the presence of collisions. Please provide pseudo-code on how you would couple

CCD with your energy function to report only closure conformations that do not contain collisions.

Note: While you are welcome to use any programming language, please typeset your deliverables. We encourage you to use Matlab. You may use any visualization software to produce your images.